Università di Pisa
Dipartimento di Ingegneria
dell'Informazione


# Monitoring POI engagement using social media and image recognition

*Candidate:*
Giulia Menichini

*Supervisors:*
Marco Avvenuti
Mario G.C.A. Cimino
Maurizio Tesconi

July 2015

# *Acknowledgements*

**My heartfelt thanks to:**

- my supervisors, that presented me this opportunity:

  Marco Avvenuti
  Mario G.C.A. Cimino
  Maurizio Tesconi

- my tutors, for their preparation and patience:

  Sergio Bianchi
  Fabio Del Vigna
  Fabrizio Falchi
  Davide Gazzé

# *Abstract*

***Social media*** is a very general term used to describe a great variety of Web-based applications and technologies, that enable people to socially interact with one another via Internet. Notable examples are Facebook, YouTube, Instagram, Twitter and so on. These applications and platforms host a notable quantity of **User-Generated Content** (**UCG**), such as photos, feelings etc. For this reason social media technologies are a precious source of information, that can be collected and analyzed to determine trends and behaviors of social media users.

In this work, we discuss the possibility of using a popular media-sharing tool like Instagram to analyze the level of appreciation (or engagement) of the **points of interest** (POI) in a city, using the photos taken by the users and their popularity by other users, expressed via "likes" and comments.

The recognition of a point of interest in a picture is performed without the help of metadata like geolocalization, but with an **image recognizer** opportunely trained to associate a point of interest with their pictures. The training process is unsupervised, meaning that our recognizer is fed a list of points of interest in a given city and photos that represent them, obtained by crawling sites that are considered a reliable source of information. The information retrieval is performed without the help of geographic metadata, but with a **tag-based approach**; instead of retrieving information relative to touristic attractions of a city via their coordinates, we collect them using a set of terms (or **tags**) that describe the city.

We developed a web application that provides access to a set of metrics regarding engagement of points of interest in a city. It provides some tools to inspect metadata and images retrieved from Social Media. The post photos are tagged by the recognizer, and available for analysis.

Two famous Italian cities have been monitored to prove that this approach is generalizable: *Florence* and *Pisa*. The obtained results show the effectiveness of this approach, but also reveal a lot of room for improvement. For this reason, we conclude this work by suggesting different ways to improve the performances of the system and more analysis that can be done on collected data.

# Contents

# List of Figures

*Ad Aurelia, Silvio, Valerio e Vanda.*

*A Lorenzo e alla mia famiglia.*

*Ai compagni di viaggio.*

*“Day by day, what you choose, what you think and what you do is who you become.“*

Heraclitus

# Chapter 1

# Outline

The description of this work is structured as follows:

- **Chapter 2 (Introduction)**: a general introduction to the concepts studied and exploited in the course of this work.
- **Chapter 3 (System structure)**: a brief description of the aim of this work and the overall structure of the system.
- **Chapter 4 (Automatic knowledge base creation)**: description of the knowledge base creation of our system, done automatically.
- **Chapter 5 (City monitor)**: description of the photo retrieval process and the recognition process.
- **Chapter 6 (The web interface)**: description of the web interface implemented to have an immediate visualization of the dataset of our system.
- **Chapter 7 (Tests results)**: presentation of tests results and some considerations about the collected data and their meaning.
- **Chapter 8 (Conclusions)**: summary of the work and of tests results.
- **Chapter 9 (Future work)**: possible future system improvements and dataset analysis.

# Chapter 2

# Introduction

In this chapter, we will see the general concepts behind the system architecture and the sources of information we dealt with.

## 2.1 User generated content (UCG)

The World Wide Web has become one of the most important sources of information for a large set of different topics. Exploiting the technological advances of Web tools, users can have access to new methods of interaction and communication via the Internet. A key feature of the so-called "participatory Web" is the content that users generate (**user-generated content, UGC**). Currently there is no universally accepted definition of UGC, but the Organisation for Economic Co-operation and Development (OECD) has proposed three main characteristics that the content created and published should possess to be considered UGC[1]:

1. The content should be published and made available over the Internet
2. The content must have a certain level of creativity
3. The content must not be created in the context of a profession.

Typically, UCG is not created by professionals, but by amateurs, without expecting a reward of any type.

[1]For a deeper understanding, http://www.oecd.org/sti/38393115.pdf

### 2.1.1 UCG and Social Media

The UGC can thus be considered the product of the use that users make of Social Media[1]. We can define *Social Media* as a group of web applications that are based on the ideological and technological foundations of Web 2.0[2], and that allow users to create, share and exchange UGC[3]. Social media usually depend on web-based technologies (both for PC and mobile) to allow individuals and communities create, discuss and modify user-generated content. They introduce substantial and pervasive changes to communication between businesses,organizations, communities, and individuals.
These changes are the focus of the emerging field of *technoself*[4][5] studies, that is, dealing with the changes of relationships between people caused by technology advancement.
Social media differ from traditional media in many ways, including quality, reach, frequency, usability, immediacy and permanence. Social media operates in a many-to-many transmission model, in contrast to traditional media that operates under a one-to-many transmission model. **Social media marketing** is defined as the process of gaining visibility and/or a more general attention through social media sites.[6]
According to the classification presented in Kaplan et Haenlein[7], social media can be divided into the following categories:

- Social Network (Twitter, Facebook, Google+, Pinterest, LinkedIn)
- Blogs (Blogger, LiveJournal, Open Diary, WordPress)
- Virtual worlds (Second Life)
- Games set in virtual worlds (Minecraft, World of Warcraft)
- Collaborative projects (Wikipedia, Wikitravel, StumbleUpon, Digg)
- Community-based content (Instagram, Flickr, YouTube, Slideshare, DeviantArt)

We will focus our attention on Social Networks, an optimal source of multimedia data generated by users.

### 2.1.2 Social networks

A social networking service can be defined as a platform to build social relations among people who share similar interests, backgrounds or real-life relationships. It usually consists of a representation of each user (often a profile), his or her social links, and a variety of additional services. Social network services are web-based and provide means for users to interact over the Internet, such as e-mail and instant messaging. Social network sites are varied and they incorporate new information and communication tools such as mobile connectivity, photo/video/sharing and blogging.

Social networks are an invaluable source of information for marketing companies, that can profit of the information made available from users through **data mining** techniques. Marketers can create customer profiles that contain customer preferences and online behavior; Facebook has been especially important to marketing strategists.

## 2.2 Social Media Engagement

By the term **Social Media Engagement** (or simply engagement[8]) we define a metric that marketers use to measure brands' effectiveness at engaging their audiences; its aim is to measure what share of audience engaged with a certain content of whatever kind, like photos, videos or free-text. **Engagement rate** represents the variation of engagement over time (days or months); it is commonly used to measure the effectiveness of certain content(s) to the audience.

Different social networks have different ways to calculate engagement rates, based on their structure and on the ways they let users express their appreciation for a particular content. Generally, the engagement rate for a certain content is defined as:

$$\frac{engagement}{reach}$$

The **engagement** term, as said before, may vary depending on the chosen web service. Here are some examples of different platforms with different engagement parameters:

- *Facebook* defines engagement as likes, comments and shares.
- *Twitter* defines engagement as @replies, retweets and mentions.
- *LinkedIn* defines engagement as the number of interactions on a post plus the clicks and followers acquired divided by the number of impressions.

The **reach** of the post can be defined as the number of people that have access to a particular content, and may or may not express their appreciation to it.

## 2.3 Media sharing

*Media sharing* consist in the publication of digital media online (photos, videos...), thus enabling a user to share them with others, publicly or with a small circle of friends. Such function is provided through both websites and applications that facilitate the upload and display of multimedial content. Such sites are often of freemium type, providing a modest amount of free storage at the beginning and more storage after the payment of a fee. Many of these websites allow you to attach the accounts of its members with those of popular social networking sites (Facebook, Twitter, etc.), allowing an even more intuitive fruition and visualization.

### 2.3.1 Photo sharing

A particular case of media sharing is *photo sharing*[9], namely the act of publishing digital photos online, sharing them with selected people or publicly[10][11][12] or even for pure storage purpose. Photo uploading and visualization is provided through websites and application.
Photo sharing usage is not limited to personal computers, but is also possible from portable devices such as phones. Some types of camera now come equipped with wireless networking and sharing functionality.
Some of the most popular photo sharing websites are Flickr[13], Imageshack, Instagram among many others. Their structures and purposes differ greatly; their datasets can be retrieved and analyzed to achieve insights on different behaviour and preferences of their users.

In the following subsection some of the most popular services of photo sharing will be analyzed.

#### 2.3.1.1 Flickr

In addition to being a popular website for photo sharing, the service is widely used for research and dissemination of the images contained in it.

The basis of a Flickr account is the *photostream*, the set of pictures uploaded by a user. Clicking an image in the photostream opens the *photopage* related to that picture, including data, comments and links for an easy sharing on social networks. Users can tag photos loaded with titles and descriptions, and images can be tagged by the owner of the image or, if the owner allows it, by other users. These text components allow an easy catalogation of images. It is also possible the georeferencing of photos of a certain album, and any album with geotagging can be shown on a map using imapflickr.
Flickr provides different levels of image sharing. A photo can be flagged as *public* or *private*: the private images are visible by default only the owner, but can be made visible to friends and / or family. *Groups* are another important tool of interaction between members of Flickr. A group can be created by any user of Flickr, which becomes the administrator and may appoint moderators. A group can be either open access or by invitation only, and the majority has its own pool of photos.
In general, many Flickr users allow the public share their photos, providing a broad set of photos tagged and classified; also other members can leave comments on any image that are allowed to view and contribute to the list of tags associated with an image.

#### 2.3.1.2 Instagram

It was created in 2010 and quickly gained popularity, with more than 100 million registered users in April 2012. It was acquired by Facebook in April 2012 for about a billion dollars. Users can upload photos and short videos, follow the feeds of other users and geolocalize images with coordinates (longitude and latitude) or the name of a place. As for the rights of content owners Instagram, the company disclaims any texts, files, images, photos, video, sounds, musical works, works

of authorship, applications, or other materials that users post on or through the service Instagram.

#### 2.3.1.3 Panoramio

Panoramio is a photo sharing service oriented to *geolocation*, owned by Google. The photos uploaded to the site can be seen directly geolocated in Google Earth and Google Maps, with new photos added to the end of each month. The goal of the site is to help users of Google Earth to learn more about a given location viewing the photos that other users have uploaded.

Panoramio asks users to organize pictures themselves using tags, which allows to easily find images relating to a particular topic, such as the city or the subject. Panoramio was one of the first sites to implement a tag cloud, so you can easily access the images tagged with the most popular keywords. The images that have (or are believed to have) as a subject people, machines, vehicles or anything inherent internal structures, or represent public events such as fairs and concerts, are excluded from visualization, as well as images potentially controversial. No allowance will be made even if the images are historical or vintage. Images deemed too creative or artistic in concept could also be excluded from Google Earth independently of other requirements met.

# Chapter 3

# System structure

## 3.1 System requirements

The purpose of our system is to monitor the engagement of the point of interest in a city[14][15] via the retrieval of photos from a popular photo sharing tool (such as Instagram[16]) and an opportunely trained recognizer.

The method that is generally used to retrieve content relative to a geographic area (such as a city or a broader area) is by accessing its geographic coordinates (if available). This is usually accomplished by creating a **bounding box** containing the area of interest, with its **center** and a **radius**; the content georeferenced in the bounding box is considered 'valid' and retrieved.

This approach has intrinsic limitations:

- geographic information may not be always available or reliable
- geolocalization may not correspond to the actual content of the information retrieved (e.g. a photo geographically set in a particular place may not portrait the actual place).
- this method is likely to discard significant content that does not fall in the bounding box

For these reasons, we chose deliberately to avoid any type of *geolocalized search* (both in knowledge creation and in the Instagram crawler) to test the amount of relevant data we could obtain; geolocalized search has the risk to collect lots of data not really relevant to our purpose, while we could try some other method that is more precise in itself, without the need to apply some type of filter to collected data.
We tested a **tag-based approach**[17] applied to data collection in every stage of the system: instead of retrieving content relative to a city via coordinates, we find a set of terms (or **tags**) that describe the type of data we are looking for (in this case, data relative to a city) and proceed to retrieve such data using that terms as search term.

To associate a photo with a point of interest, we studied in particular the case of Tripbuilder[18]; in this study they associate a photo to a PoI when the photo was taken within a circle having the PoI as its center and r = 100 meters as radius. Figure 3.1 shows a visual example of the approach.

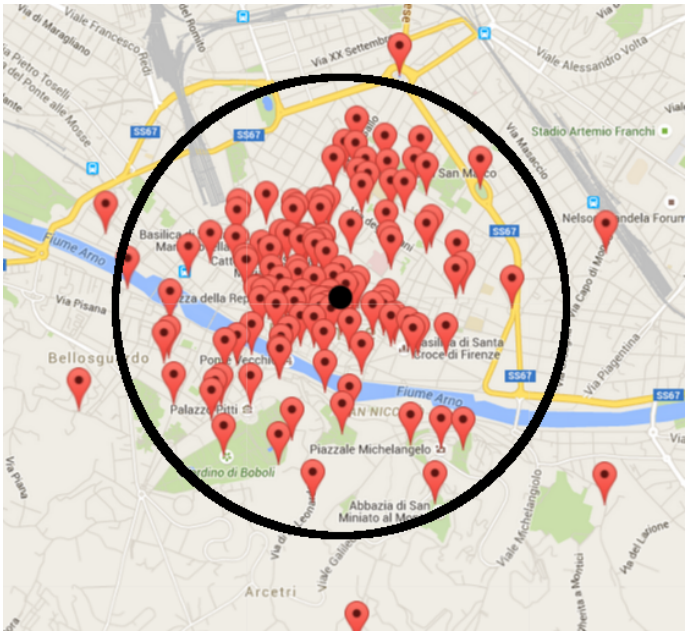


FIGURE 3.1: Geolocation retrieval

In our case, since we wanted to avoid entirely georeferenced information, we will exploit a **recognizer** tool, that must possess knowledge of the points of interest and the photos regarding such points of interest.
The knowledge retrieval process is done in an unsupervised way; it is based exclusively on the UCG found on the Internet in sites considered a source of *reliable*

information. The practical retrieval of the data is done via **Application Programming Interfaces** (commonly called **API**[19]), with which websites make available their content to programmers.

In this prototype, we performed an extensive crawling of the Italian cities of *Florence* as example, and a reduced[1] crawling of the city of *Pisa* to demonstrate that this approach can be used on any city.

### 3.1.1 Non-functional requirements

In the following, we will present some of the non-functional requirements that our system has to fulfill, regarding the performances of the system rather than its specific behaviors.

- **Usability**: the interaction with the systems must be kept to a minimum: a system administrator must only insert the name of the city to start collecting data, while a user that wants to see the collected data must use an interface easy to consult and comprehend.
- **Recoverability**: the system depends entirely on API and Web services, so a temporary failure on their part must not cause the system to stop working entirely
- **Security**: any malicious user that may want to alter data must not have access to it.

## 3.2 System schema

The system is divided into three main components:

- **automated knowledge creator**
- **Instagram crawler**
- **recognizer** (as external tool)

[1] because of traffic limits that will be explained in Chapter 5

A **web interface** was developed too, to consent a graphical visualization of our data by means of various types of graphs and maps.
Each single component will be analyzed and their funcionalities explained in the following chapters. In figure 3.2 a general schema of the system is presented.

### 3.2.1 Tools

The system runs on a *VMware Virtual Machine* with *Ubuntu*[2] *14.04.2 LTS* (release 14.04) as OS, hosted at `http://wafi.iit.cnr.it/poi/poi/` with *RAM size 4GiB* and four *AMD Opteron(tm) Processor 4332 HE*.
In developement, we used popular web scripting languages such as **PHP**[3] and **Javascript**[4]; the recognizer tool was written in **Java**[5], and some interfaces of it were re-implemented to improve performances. The data collected was stored in a MySQL[6] database to allow an easy retrieval, accessed with phpMyAdmin[7].

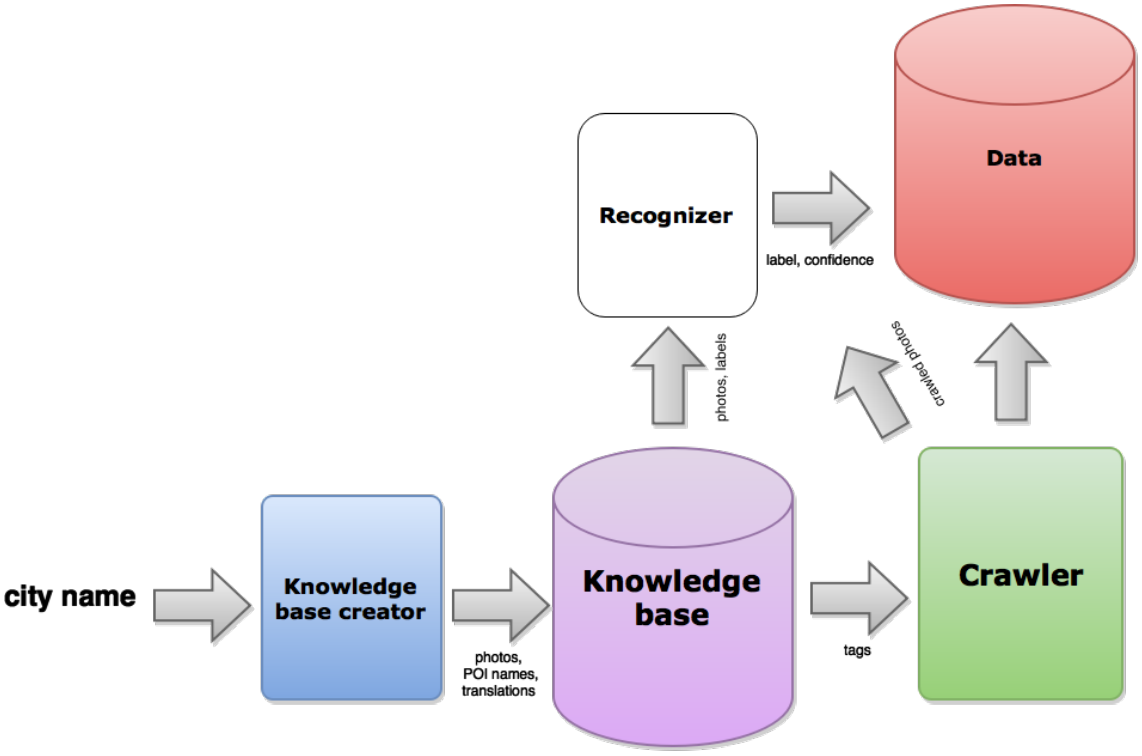


FIGURE 3.2: General schema of the system

[2] `http://www.ubuntu.com/`
[3] `http://www.w3schools.com/php/`
[4] `http://www.w3schools.com/js/`
[5] `https://www.oracle.com/java/index.html`
[6] `https://dev.mysql.com/`
[7] `https://www.phpmyadmin.net/`

## 3.3 Deployment section

Each component is composed of a set of files that determine their working:

### 3.3.1 Automated knowledge creator

This module is composed of:

- **sparqllib.php**: a PHP interface to query DBPedia dataset
- **POI_extractor.php**: retrieves a list of points of interest from the chosen source
- **wikicommons_dw.php**: retrieves a list of images relative to the points of interest from the chosen source
- **trainer.php**: this file interfaces with the Java module to train the recognizer with the knowledge built in the previous stages

### 3.3.2 Instagram crawler

This module is composed of:

- **tagsearch.php**: this file retrieves a list of tags from Instagram, given a set of translations of the city name
- **crawler.php**: this file retrieves a list of media content from Instagram
- **engagement.php**: this file retrieves engagement metadata relative to the posts from Instagram
- **followers.php**: this file retrieves user information from Instagram
- **multi_recognizer.php**: this file sends batch of photos to the Java recognizer that proceeds to analyze and recognize points of interest in them

### 3.3.3 Recognizer

This module is composed of:

- **VLADKnowledgeBaseAdd.jar**: provides an interface to add photos to the knowledge base
- **VLADSearch.jar**: provides an interface to perform the recognition process

In addition to the files forming the core functionalities of the system, a series of utilities were implemented:

- **db_utilities.php**: a series of interfaces to the database.
- **multiRequest.php**: a tool to allow multiple download at once, particularly useful for the recognition process.
- **imageconversion.php**: because the Java recognizer only processes JPEG images, this file converts images of any format to such standard.

# Chapter 4

# Automatic knowledge base creation

In this chapter, the knowledge base building process is described in detail.

The knowledge base of a given city consists of:

- a **points of interest** list
- a collection of **images** representing such points of interest
- a list of **translations** of the city name in different languages used for an extensive crawling

The goal of this stage is to build such knowledge collecting the necessary information exclusively via **web crawling**; the chosen sources of information are Wikipedia[1] for the retrieval of points of interest information and Wikimedia Commons[2] to build the image collections.

## 4.1 Module structure

In figure 4.1 is displayed a diagram of the knowledge base building process.

[1] https://en.wikipedia.org/wiki/Wikipedia:About
[2] https://commons.wikimedia.org/wiki/Commons:Welcome

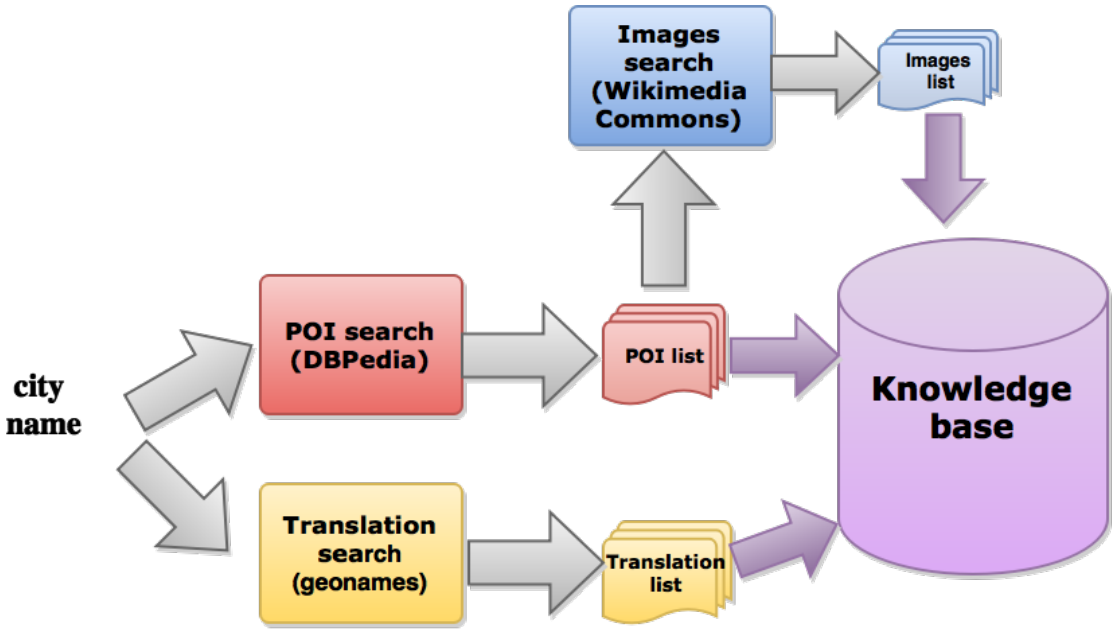


FIGURE 4.1: Knowledge base builder structure

## 4.2 DBPedia

DBpedia[20] (from "DB" for "database") is a project that aims to extract information created as part of the Wikipedia project in a structured way[21]. It allows users to get relations and properties related to the resources of Wikipedia, including links to other related data sets. It is made available under free licenses.

Wikipedia articles consist mostly of text, but also include structured information such as tables, categories, images, geographic coordinates and links to external websites. Such structured information are included in a set of uniform data that can be interrogated, retrieved and analyzed.

The DBpedia project relies heavily on the concept of Semantic Web[22]. It uses the Resource Description Framework (RDF)[23] to represent information and consists of 3 billion RDF triples, 580 million drawn from the English edition of Wikipedia and 2.46 billion of other language editions.

There are various ways to query the DBPedia dataset: via various DBPedia endpoints (like `http://dbpedia.org/sparql`, the official one) or as Linked Data, accessible through a series of Semantic Web browsers (DISCO, Marbles, the OpenLink Data Explorer...).

## 4.3 Points of interest retrieval

Instead of exploiting the commonly used geolocalization retrieval, the system performs a *category crawling* of the DBPedia Dataset (which can be viewed as a graph) regarding touristic informations on a given city. The retrieval was performed with the help of the *SPARQL PHP library*[3], freely available for use.

The crawler starts from a macrocategory about visitor attractions in the chosen city and then explores its subcategories, collecting Wikipedia pages that belong to each visited category. The process is *recursive*: every subcategory may have its own subcategories, and so on.

To retrieve the necessary informations from DBPedia, a set of so-called *semantic queries* are sent to the endpoint. Generally speaking, the structure of a **semantic query** differs in some aspects from the standard queries, in line with the structure of Semantic Web. Semantic queries are based on the concept of **triple**, a data entity composed of *subject-predicate-object*. A typical semantic query requires at least:

- a **graph pattern**: one or more triple patterns contained within curly braces ( ). It specifies the range of our query.
- a **result clause**, that specifies the information to return from the graph. There are several types of result clauses, although the most commonly used is the SELECT statement.

Variables are generally denoted with a question mark (?). In a graph pattern they match any node - whether resource or literal.
In the following snippet, we see the query used to retrieve all the pages that fall into the specific category regarding visitor attractions in Florence.

```
prefix dc: <http://purl.org/dc/terms/>
prefix skos: <http://www.w3.org/2004/02/skos/core#>
prefix category: <http://dbpedia.org/resource/Category:>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>


SELECT DISTINCT ?name ?image ?lat ?long ?wikipage
```

[3] http://graphite.ecs.soton.ac.uk/sparqllib/

```
where {

        ?s dc:subject category:Visitor_attractions_in_Florence
        ?s rdfs:label ?name .
        ?s foaf:depiction ?image .
        ?s geo:lat ?lat .
        ?s geo:long ?long  .
        ?s foaf:isPrimaryTopicOf ?wikipage
        FILTER (langMatches(lang(?name), 'en'))
        }
```

In this query we have a graph pattern composed of 6 triples, and the return clause was of type SELECT (that indicates that you are requesting data from a dataset). The FILTER keywords discards all the results that are written in a language different from English.

The PREFIX clause defines prefixes and namespaces, for abbreviating URIs representing predicates.

This query retrieves only **geographical entities** (like building, famous squares or statues). From the result of such query we retrieve the following informations from DBPedia:

- **name** of the POI
- **thumbnail** of the Wikipedia page of the POI
- **latitude** and **longitude** of the POI
- **Wikipedia page** of the POI.

During the crawling process, since every page could belong to multiple categories (making the dataset more a *network* than a tree) there is the possibility of duplicates in raw results, that are however eliminated programmatically. Figure 4.2 shows an example of this: the red arrows show the execution of the algorithm in the left subgraph, with page 2 as duplicate in the result. The circles represents the pages to retrieve (the *edges* of the category graph).

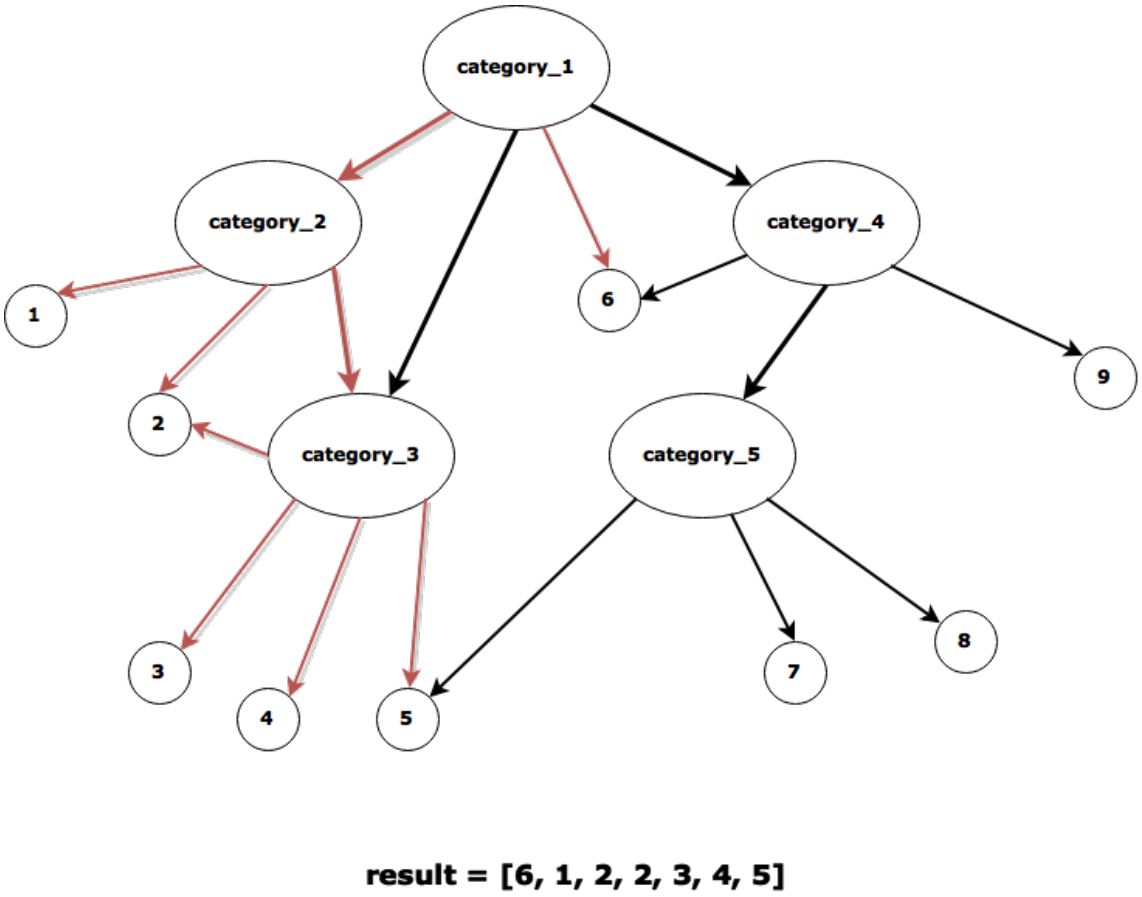


FIGURE 4.2: Example of a case of duplicated results

In the following snippet, we see the query used to retrieve all the subcategory of a specific category, while in figure 4.3 the schema of the POI retrieval process is presented.

```
SELECT DISTINCT ?subject
                where {

                {
                        ?subject skos:broader category .
                }
```

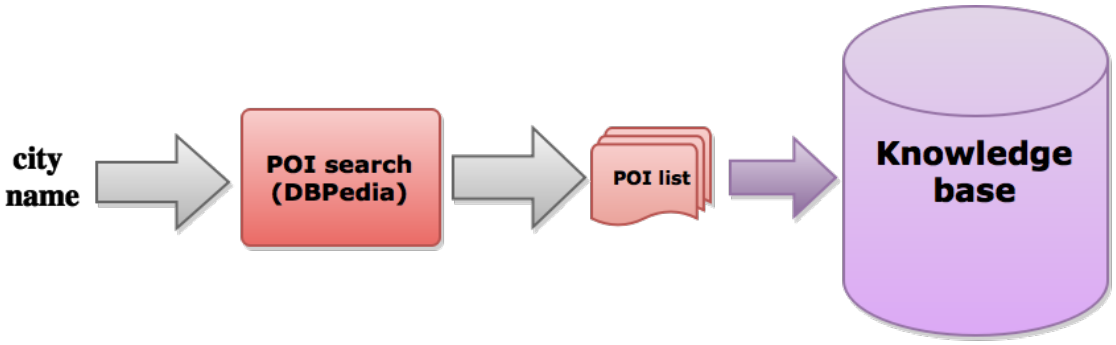


FIGURE 4.3: POI Retrieval

## 4.4 Images retrieval

The crawler interrogates the Wikimedia Commons API to get a set of images regarding the points of interests retrieved previously from DBPedia. The API is called **MediaWiki**[4], and it consists of free server-based software which is licensed under the GNU General Public License (GPL). MediaWiki is an extremely powerful, scalable software and uses PHP to process and display data stored in a database, such as MySQL.
Unfortunately, at the best of our knowledge, the MediaWiki is not directly connected in any way to the DBPedia API, so a bit of text processing is needed to retrieve the correct names to use as categories of the images. For this purpose we used the lynx[5] tool.
The API is able to return data encoded in various standards; we chose the **JSON**[6] format because it is extremely easy to retrieve information from this format, particularly in Web languages such as PHP (via the `json_encode`and `json_decode`functions).

The following is an example of API request: `http://commons.wikimedia.org/w/api.php?action=query&format=json&generator=categorymembers&gcmtype=file&gcmtitle=Category:Ponte_Vecchio&prop=imageinfo&&iiprop=url`

From this response we extract the *url* information from each group `imageinfo` and we store it in our image database along with the label of the associated point of interest.

```
" imageinfo ": [
{
```

[4] https://www.mediawiki.org/wiki/MediaWiki
[5] http://linuxcommand.org/man_pages/lynx1.html
[6] http://json.org/

```
" url ": " https :// upload . wikimedia . org
/ wikipedia / commons /4/42/03_2015 _Ponte_Vecchio - Arno -
Portico - ordine_tuscanico%2C_arco_a_tutto_sesto_ %28Firenze%29 _Photo_Paolo_Villa_FOTO 9247bis
34 " descriptionurl ": " https :// commons .
wikimedia . org / wiki / File :03_2015 _Ponte_Vecchio - Arno -
Portico - ordine_tuscanico , _arco_a_tutto_sesto_ ( Firenze )
_Photo_Paolo_Villa_FOTO 9247bis . JPG "
 }
]
```


The url is subsequently processed to download the image to train the recognizer, that will act as a classifier.

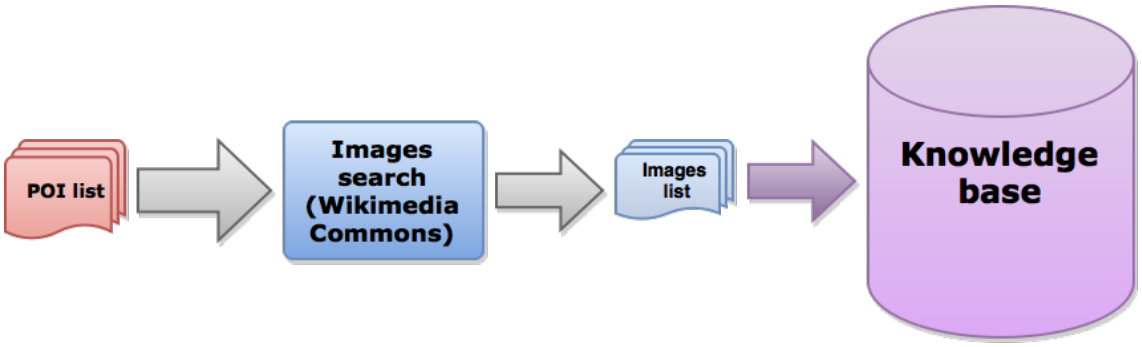


FIGURE 4.4: Images retrieval

## 4.5 Translation retrieval

For a broader retrieval of data from our source, we need to obtain the translation of the city name in different languages: this is possible thanks to **geonames.org**, a worldwide geographical database[7] with a search function and downloadable data files. From one of these files we managed to collect the translation of 'Florence' in various languages, included eastern languages like Japanese and Korean.
This procedure was however applied only to the city of *Florence* to avoid exceeding the strict limits of the Instagram API (see the relative section).

[7] http://www.geonames.org/

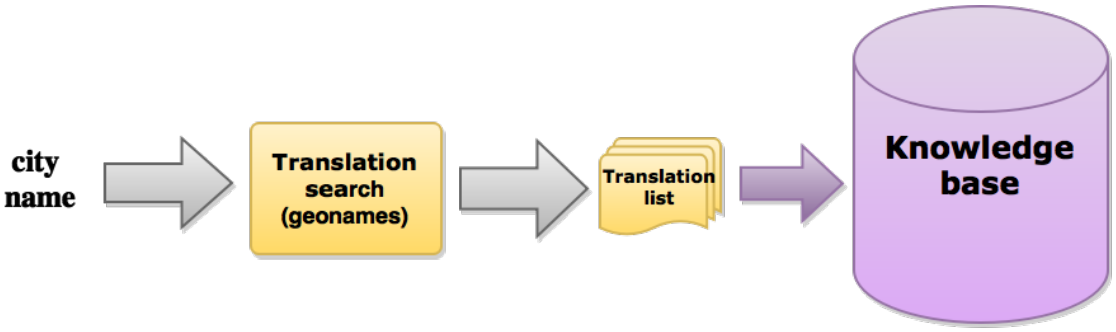


FIGURE 4.5: Translation retrieval

## 4.6 Recognizer training

Once we have the list of points of interest of a city and their image representation, we can use them to train a recognizer that will perform the recognition process. The process of recognition of the specific points of interest in our photos is performed by an external tool developed in Java; such tool must first be trained with a set of images that represents the knowledge base of the system. The urls of the images are retrieved from the database, then the pictures are downloaded in parallel using the `curl_multi`[8] methods, that allows for multiple download in parallel.

The recognizer is given a set of training samples to acquire the knowledge base necessary to perform a correct recognition. Formally, a training sample can be defined as a couple of values:

```
<photo,label>
```

The recognizer extract **VLAD** and **ORB**[24] features from the photos, and saves them in a *dictionary* (stored in a file) along with the associated labels and a unique id. Such dictionary will be used as a *reference* for the recognition process.

### 4.6.1 Java interface

The training interface of the system is the following:

[8] http://php.net/manual/en/function.curl-multi-init.php

```
public void add(File imageF, IDString id, StringLabel label);
```

This function adds the image `imageF` with label `StringLabel` and id `IDString` to the dictionary.

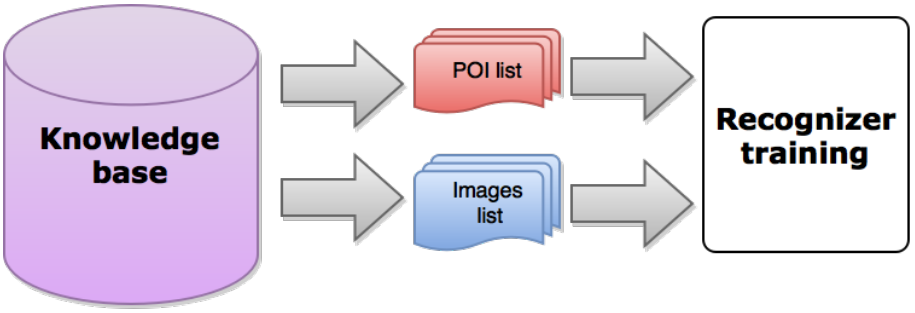


FIGURE 4.6: Training process

## 4.7 Results

For the city of **Florence**, the automatic knowledge base creator retrieved 147 points of interest and 895 images, while for **Pisa** we obtained 43 points of interest and 286 photos.

The method reduces the risk of junk results, but is likely to discard some places of interest that don't fall into the right categories because of human errors or missing links. For example, this method did not allow the retrieval of specific visitor attractions, because they belong to the wrong categories, and thus they are unreachable by our crawler. Donatello's *Perseus with the head of medusa*[9]), for instance, belongs only to the categories *Sculptures in Italy* and *Renaissance sculptures*, and none of the two are subcategories of visitor attractions of Florence.

This issue could be resolved considering that for an Italian city it is better to retrieve POI data from the Italian DBpedia, which contains more precise informations about Italian cities in its dataset; the same reasoning can be applied to cities belonging to different countries (e.g. German DBPedia for Berlin, French DBPedia for Paris...).
DBPedia datasets of different languages are however based on different ontologies, which makes harder to achieve global compatibility, and there can be some property mismatch between names or values in different languages.

[9] http://dbpedia.org/page/Perseus_with_the_Head_of_Medusa

# Chapter 5

# City monitor

In this chapter, the photo crawling process is described in detail.

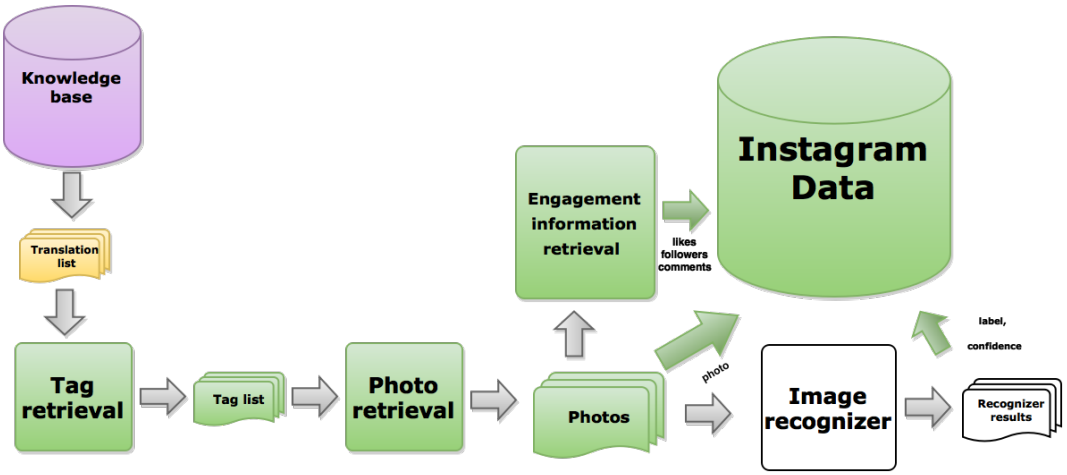


FIGURE 5.1: Crawler structure

For this prototype, **Instagram** has been chosen as our photo source; the crawler periodically polls the website for posts annotated with a certain **tag** and retrieves the data associated with such posts.

## 5.1 Tags

Generally speaking, a **tag** is a keyword or term assigned to a piece of information on the Internet (an image, or computer file). It helps describe an item[25] and allows it to be found again by browsing or searching. Tags are generally chosen personally by the content's creator.

Tags were popularized by websites associated with Web 2.0 and are an important feature of many Web 2.0 services.

## 5.2 Instagram API

Instagram provides access to their data via a set of endpoints accessible via https, located at `api.instagram.com`.

For instance, this endpoint allows the user to retrieve photos with the hashtag "nofilter" by accessing the following URL with a valid **client ID**: `https://api.instagram.com/v1/tags/nofilter/media/recent?client_id=CLIENT-ID`.

Some API only require the use of a client id that associates a server, script, or program with a specific application; other requests require an authentication, specifically requests made on behalf of a user, such as POST requests. Authenticated requests require an **access token**.

To avoid excessive load on servers, Instagram APIs apply a global rate limit of calls performed by a valid client-id over an 1-hour sliding window, regardless of the endpoint(s) used. Information regarding the global rate limits is included in the HTTP header on the response to each call, to easily determine the status of the rate limits.
In our application we had at our disposal two client ids, thus raising the rate limit to 10,000 requests per hour; but even with this advantage the limit was barely sufficient for our needs; sudden peaks of traffic can slow down the performances of the system.

Instagram API gives back a **JSON** response, that can be processed in most web languages (PHP and Javascript above all) to extract the information needed. All the information extracted from API requests is subsequently saved in a database, to allow an easy retrieval.

## 5.3 Post crawling

The Instagram crawler consists in a series of cron[1] jobs[2]; each of them periodically interrogates different endpoints of the Instagram API to retrieve the necessary informations.

A set of tags containing the city name is retrieved from this endpoint: `https://api.instagram.com/v1/tags/search?q={word}&client_id=CLIENT-ID`

An example: `https://api.instagram.com/v1/tags/search?q=firenze&client_id=01bcd43bbfa4421ebf5734f00397e1db`

From the response, we extract the tag name from the `name`field.

```
1
2 {
3 " media_count ": 3938,
4 " name ": " firenze2014"
5 },
```

This procedure is applied to every translation of the city name stored in our knowledge base.

### 5.3.1 Post retrieval

Posts associated with a certain tag are retrieved via the *tag endpoint*: `https://api.instagram.com/v1/tags/{tag-name}/media/recent&client_id=CLIENT_ID`

An example: `https://api.instagram.com/v1/tags/florence/media/recent?count=33&client_id=83afcd79589440e0a4f5e8eba16efe79` returns the following JSON response (here there is a snippet as example):

This request takes as parameter the tag to monitor (`florence` in this particular case) and gives back a JSON response (listed in Appendix).

From the response, the following relevant information is extracted:

- **id** of the media content: this data is found as the `id`field.

[1] http://ss64.com/bash/cron.html
[2] The transcription of the used crontab file is found in the Appendix A

```
" id ": "1024613840405115709_712964843"
```

- **URL of the photo** associated with the content: this data is found as the `url`field, in the `standard_resolution`format.

```
" standard_resolution ": {
 " url ": " https :// scontent . cdninstagram .
com / hphotos - xfa1/ t51.2885-15/ s640x640/ e35/ sh0.08/1135
   6793
_1616653911905886_2110550807_n . jpg ",
 " width ": 640,
 " height ": 640
}
```

- **id** of the owner of the content: this data is found as the `id`field in the `from`group.

```
" from ": {
 " username ": " marievandeweyer_ ",
 " profile_picture ": " https ://
instagramimages - a . akamaihd . net / profiles / profile_ 71
   2964843
_75sq_1385398269. jpg ",
 " id ": "712964843",
 " full_name ": " Marie Vandeweyer "
},
```

- **tags** associated with the content: these data are found under the `tags` array.

```
" tags ": [
 " tuscany ",
 " panorama ",
 " cityhopping ",
 " firenze ",
 " duomo ",
 " italy ",
 " florence ",
 " view "
 ]
```

- **time of creation** of the content: this data is found (in *seconds*) as the field `created_time` .

```
" created_time ": "1436363509",
```

- latitude and longitude of the photo (if geolocalized): these data are found as `latitude` and `longitude` under the group `location`.

```
" location ": {
 " latitude ": 43.762958184,
 " name ": " Forte di Belvedere ",
 " longitude ": 11.256113019,
 " id ": 289038127
 }
```

Every API call returns a maximum of 33 photos; to get the next round of photos in chronological order we have to retrieve the field *"next url"*, that gives the next set of the most recent 33 photos in chronological order. The crawling session stops when we accumulate a certain number of photos with a timestamp that preceed the last timestamp for that tag from our database. Figure 5.2 illustrates the structure of the crawling.

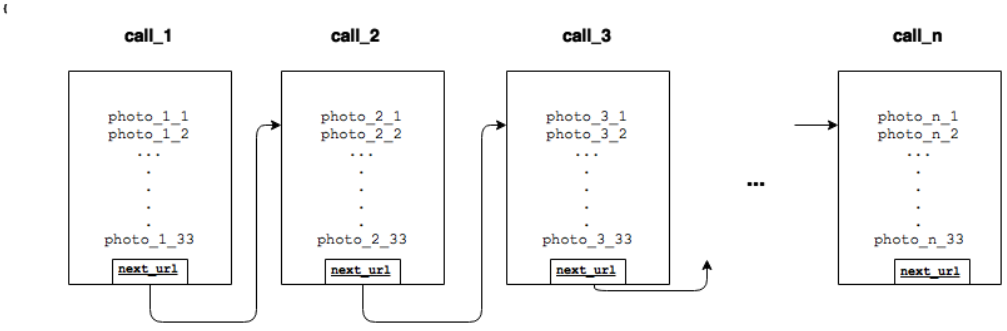


FIGURE 5.2: Instagram post retrieval

Ex. `#florence` → `last_timestamp`: 06-07-2015 18:09:04
Once the crawler found 33 photos created before `last_timestamp` we stop our retrieval for that session.

### 5.3.2 Engagement data retrieval

The engagement information (i.e. likes and comments) is not retrieved immediately, but after a certain period of time (typically 6 hours), when a good number

of likes and comments has been collected for that post. Engagement information of the crawled posts is retrieved using the following media endpoints:https://api.instagram.com/v1/media/{media-id}?access_token=ACCESS-TOKEN

It is necessary to use a different endpoint to retrieve these data because the information provided by the endpoint interrogated before is not reliable: the number of likes and comments displayed there is limited (for instance, up to a maximum of 120 likes is returned).

An example: https://api.instagram.com/v1/media/1028951906564929504?client_id=83afcd79589440e0a4f5e8eba16efe79

This request takes as parameter the media-id (1028951906564929504 in this particular case) and gives back a JSON response (listed in Appendix).

From the API response, the following relevant information is extracted:

- number of likes of the photo: this data is found as the field `count` in the group `comments`.

```
" comments ": {
 count ": 0,
 data ": []
},
```

- number of comments posted: this data is found as the field `countin` the group `likes`.

```
" likes ": {
" count ": 1,
" data ": [
 {
" username ": " carachamber 123",
" profile_picture ": " https :// igcdn - photos
-e - a . akamaihd . net / hphotos - ak - xaf1/ t51.2885-19/ s
    150x150/11
8356822_682674911876156_864718820_a . jpg ",
" id ": "1387869787",
" full_name ": ""
}
]
}
```

### 5.3.3 User information retrieval

To retrieve the number of followers of a certain user, we use the following media endpoint:

https://api.instagram.com/v1/users/{user_id}/?client_id={client_id}

Example: https://api.instagram.com/v1/users/1264687840/?client_id=83afcd79589440e0a4f5e8eba16efe79

This request takes as parameter the user-id (`1264687840` in this particular case) and gives back a JSON response (listed in Appendix). From its JSON response, we can find the number of followers of a given user in the field `followed_by`in the group `counts`.

```
" counts ": {
" media ": 1185,
" followed_by ": 362,
" follows ": 322
},
```

## 5.4 Image recognition

The crawled images are periodically retrieved from the database and subject to recognition using the external Java tool mentioned before. When given a photo, the recognizer extracts VLAD and ORB[24] features from it and confronts them with its dictionary of `<photo,label>`couples. Each city has its own dictionary file.

Each dictionary has a `VLADEngine.properties` file associated, that contains, among other things, the name of the archive(s) in which the dictionary is stored and the name of the working directory.

### 5.4.1 Java interface

The recognizer interface of the system is the following:

```
public static SearchResult search(Properties prop, File imageF);
```

This function searches for the properties file named `prop` , processes with the associated dictionary and returns a result that consists of a structure of type `SearchResult`.
Two outputs are possible:

- **success**: the recognizer has found a photo that considers similar to the input photo: in this case, the following fields are returned:
  - the **label** associated to said photo
  - the **id** of the most close dictionary photo
  - the **confidence** of the recognition, a real number between 0 and 1 that defines the *degree of certainty* of the recognition process for that particular photo.

  The database is then updated with the name of the point of interest (if found) and the confidence value.
- **failure**: the recognizer hasn't associated the image `imageF` to any of the photos in its knowledge pool; in this case, it returns only the string `null` .

To speed up the recognizing process, Java multithreading was exploited. We couldn't launch multiple instances of the recognizer, since the dictionary was loaded in the JVM for every instance and its size (multiplied by the number of istances of the recognizer) filled the Java environement very quickly.
A set of photos is downloaded and stored in filesystem, and the system recognizes every photo in a separated thread, using the dictionary as a shared only-reading object, that in this way is loaded only once in the heap in the JVM. Figure 5.4 shows the structure of the threads.

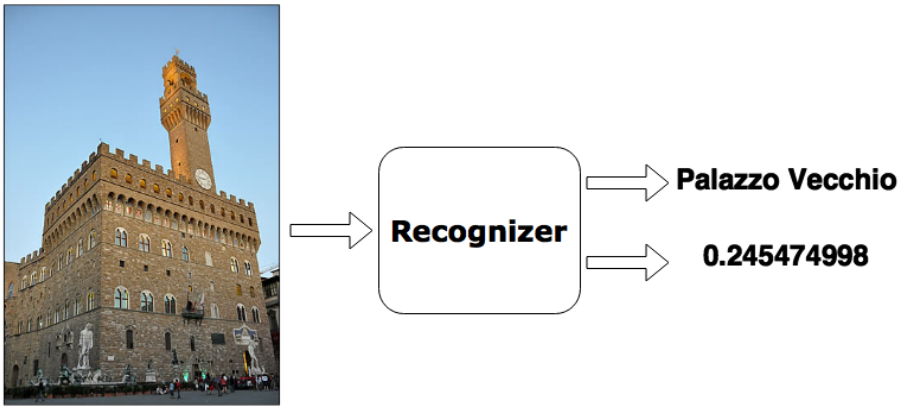


FIGURE 5.3: Example of recognition process

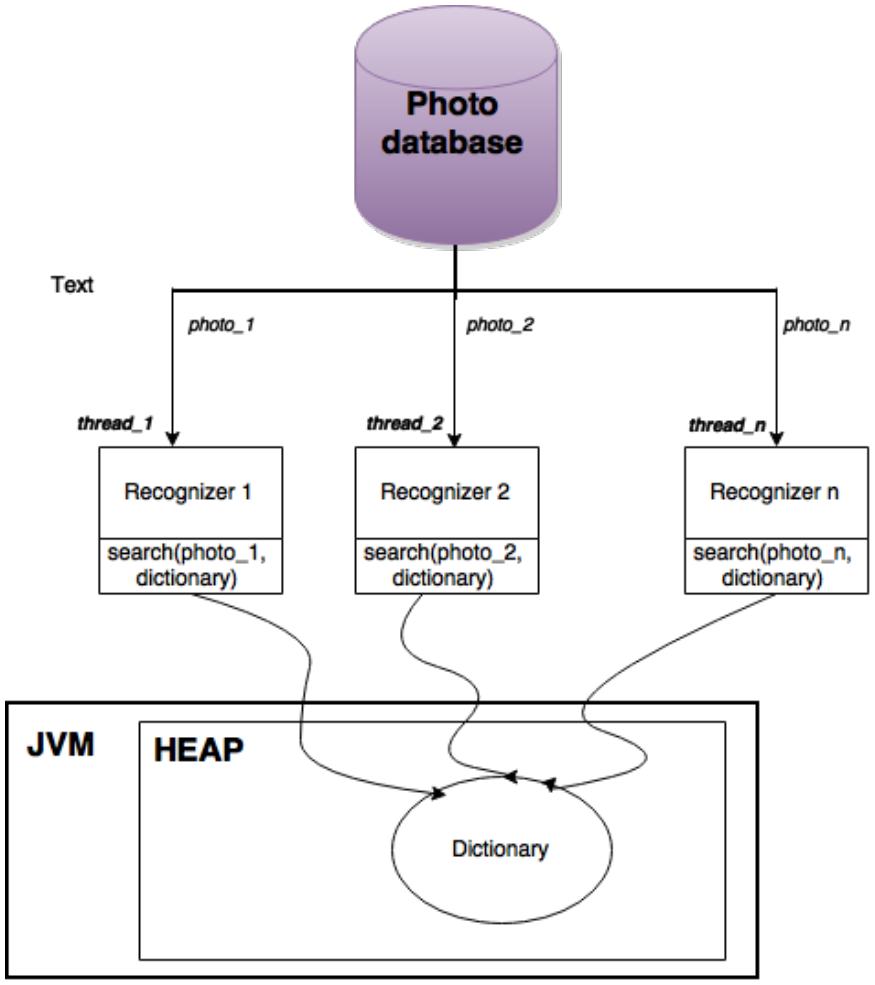


FIGURE 5.4: Java multithreading for performance improvement

## 5.5 Engagement rate calculation

So far, along with the informations regarding specific photos and their points of interest, the tool is keeping trace of the engagement informations associated with each photo:

- **likes** of the given photo
- **comments** on the given photo
- **followers** of the user who took the photo

Every photo taken by a certain user generates its own engagement, that is defined, as said before, by $\frac{total\_engagement}{user\_reach}$. In this particular case, we can define the **total**

**engagement of a certain post** as the number of likes plus the number of comments of each photo; the **reach** can instead be defined by the number of followers of the user that published the photo. Then all the contribution of the users are added to obtain the engagement rate of that specific point of interest.

So, a measurement of the engagement of a point of interest can be formalized in the following way:
$\forall\, p_i \; in \; P : p_i \; is \; a \; POI$
$\forall\, u_j \; in \; U : u_j \; is \; an \; user$
$\forall\, f_k \; in \; F : f_k \; is \; a \; photo$
$U = users \; who \; took \; photos \; of \; p_i$
$F = photos \; of \; p_i \; taken \; by \; u_j$
$|U| = n \;\; |F| = m$

$$e_{p_i} = \sum_{u_j=1}^{n} \frac{\sum_{f_k=1}^{m} (likes_{f_k} + comments_{f_k})}{followers_{u_j}}$$

First, for each user, $e_{u_j}$ is calculated (that is, the engagement for a POI generated from that particular user); then the global engagement rate ($e_{p_i}$ is calculated as the sum of the individual engagement rates of all users.

This is not the actual formula we used to display data in our web interface: we display only a crude sum of likes and comments of a given POI, without dividing it for the number of followers, because we wanted to visualize a simple measurement of the appreciation for a particular point of interest, without any penalization given by normalization.

# Chapter 6

# The web interface

To allow an easy observation of the data collected, a web interface has been implemented, that allows us to visualize some interesting statistics that may emerge from our dataset. The interface is available at http://wafi.iit.cnr.it/poi/poi/.

In the sidebar on the left, two datepickers (generated with the help of the jQuery[1] plugin) allow an intuitive selection of the dates between which the user wants to see the data. In the same sidebar, a select allows to choose the city whose data we want to see, and some statistics regarding the recognizer are displayed. The charts are shown in the central panel, with some tab that allow to choose the type of chart.

## 6.1 Map of the city

Using the Google Maps API[2] a map of the selected city is displayed, with the points of interest (retrieved from our knowledge base) displayed as markers (Figure 6.1). The markers are encoded in XML and passed to the Google Maps API in such format. When the user clicks on a marker, the name of the point of interest and a photo of it are displayed.

[1] https://jquery.com/
[2] https://developers.google.com/maps/

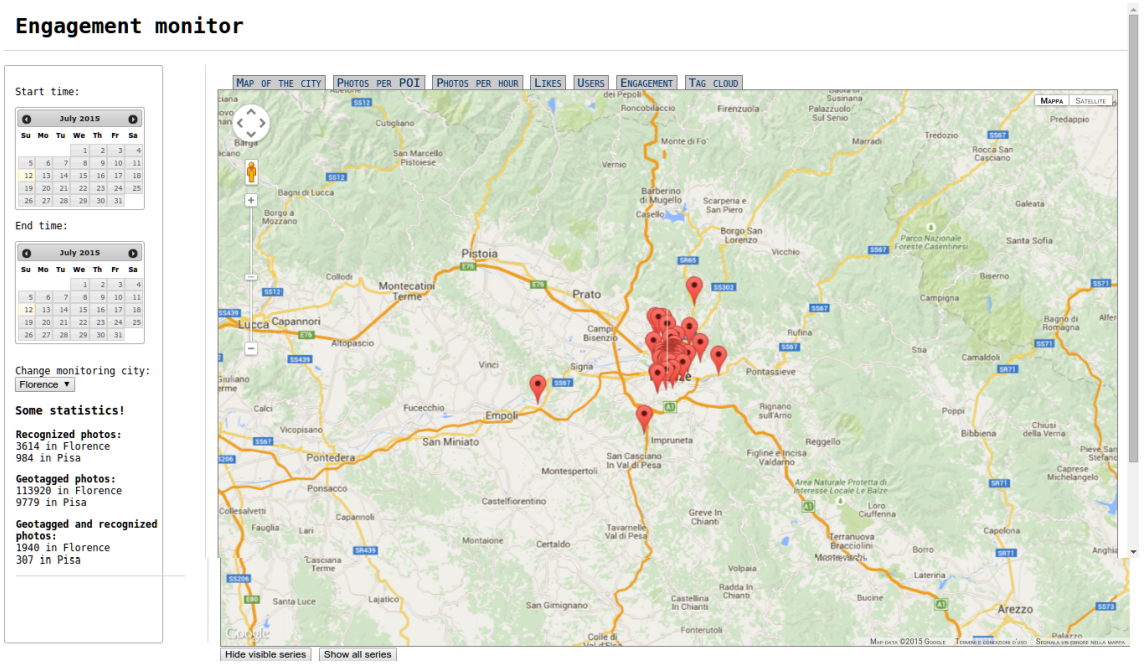


FIGURE 6.1: Map interface

## 6.2 Photos per POI

This chart (made with Highcharts[3], a JQuery plugin, as most of the charts here) displays the amount of photos per point of interest across the chosen days ((Figure 6.2).

Each series represents a specific point of interest; a legend displaying the name of each series is displayed on the left of the graph.

## 6.3 Photos per hour

This chart displays the amount of photos recognized across the hours in different days (Figure 6.3). Here each series represents a different day, as reported in the legend.

[3] http://www.highcharts.com/

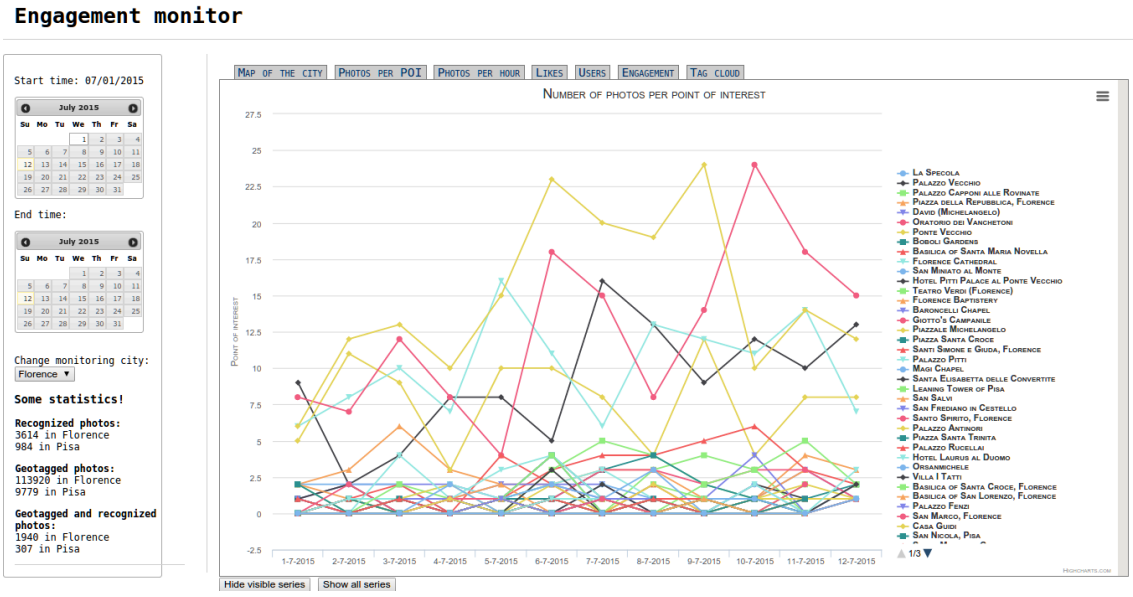


FIGURE 6.2: Photos per POI

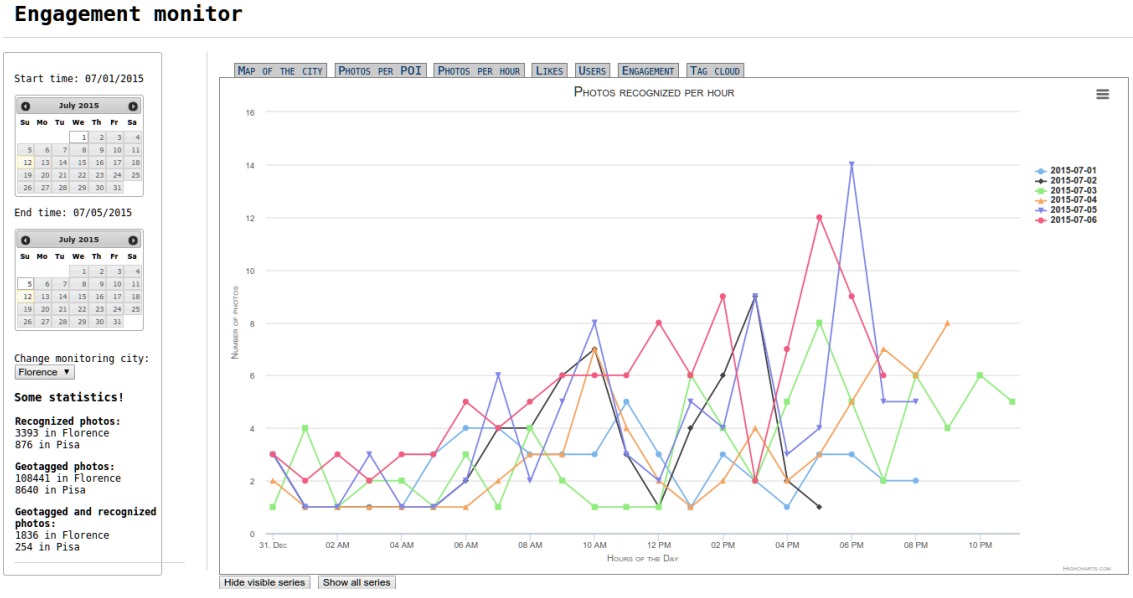


FIGURE 6.3: Photos per hour

## 6.4 Likes

This chart displays the sum of likes for each point of interest (Figure 6.4) Here each series represents a point of interest, like in the first chart, as reported in the legend.

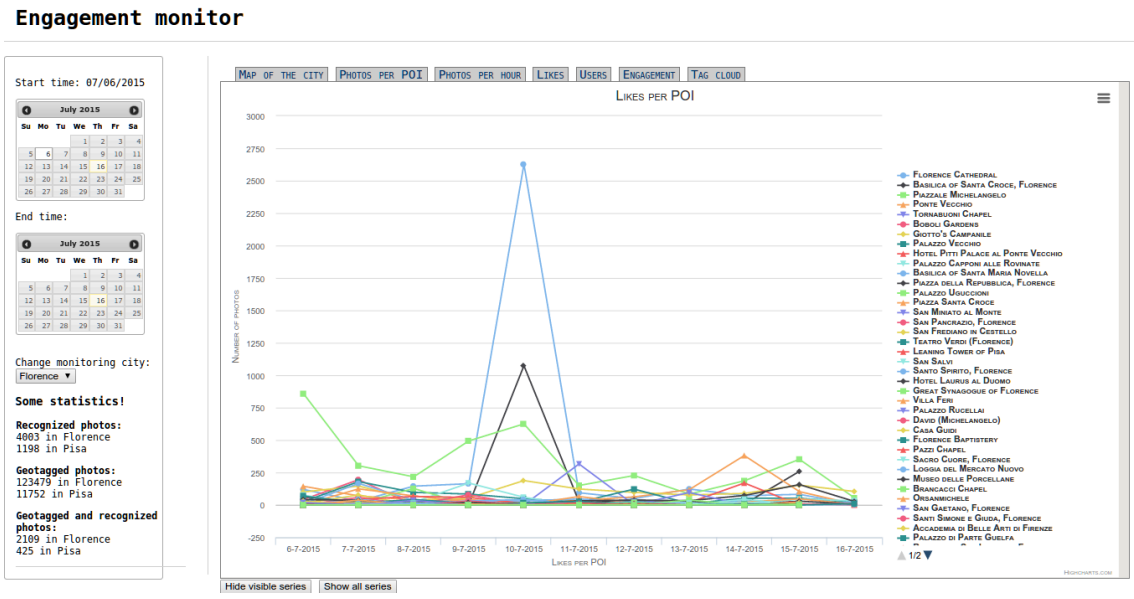


FIGURE 6.4: Sum of likes for each point of interest

## 6.5 Users

This chart displays the number of recognized photos taken by individual users over time[4](Figure 6.5). it consists of a stacked column diagram, to easily visualize the different contributions of the users and at the same time have a visual representation of the amount of users for each day.

## 6.6 Engagement

This chart displays the engagement rate for each point of interest, calculated, as said before, as a crude sum of likes and comments (Figure 6.6) It is worth noting that this chart differs very little from the *Likes* chart; this observation will be investigated further in the following chapter.

## 6.7 Tag cloud

This chart represents the frequency of tags retrieved in our dataset: the font size of each word represents its frequency relative to other tags.

[4]This chart unfortunately has a very slow loading time for long periods, because of the largeness of data that slows down the chart library.


FIGURE 6.5: Relevant photos taken by each user

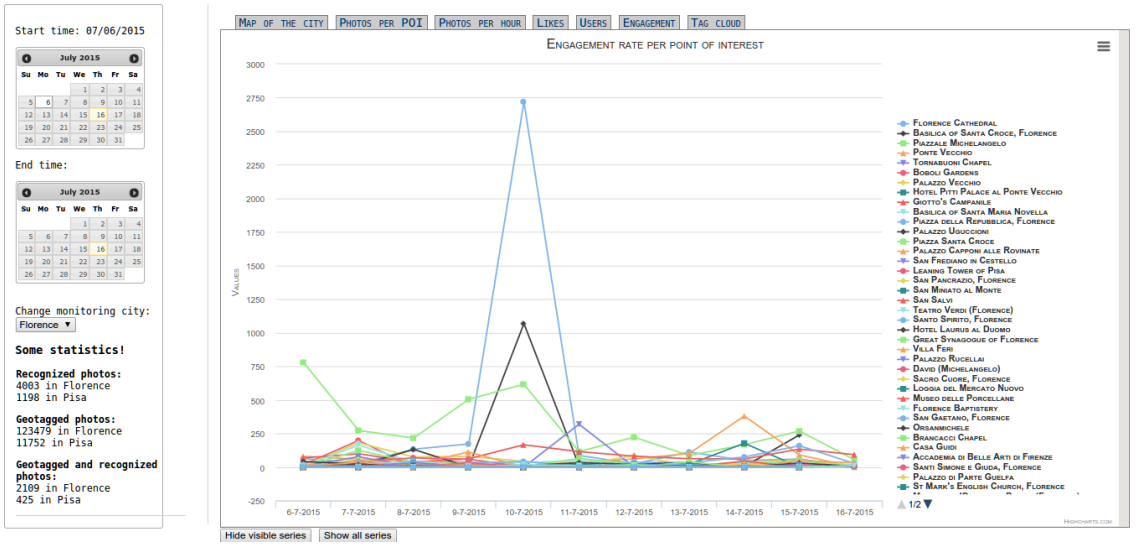


FIGURE 6.6: Engagement rate per point of interest

It is generated via the d3 algorithm[5] for tag clouds. Like the other figures, the tag cloud isn't just relative to one day but can cover a chosen period of time, allowing various considerations about the frequency of the tags in it (Figure 6.7).

[5]An explanation here: https://www.jasondavies.com/wordcloud/about/

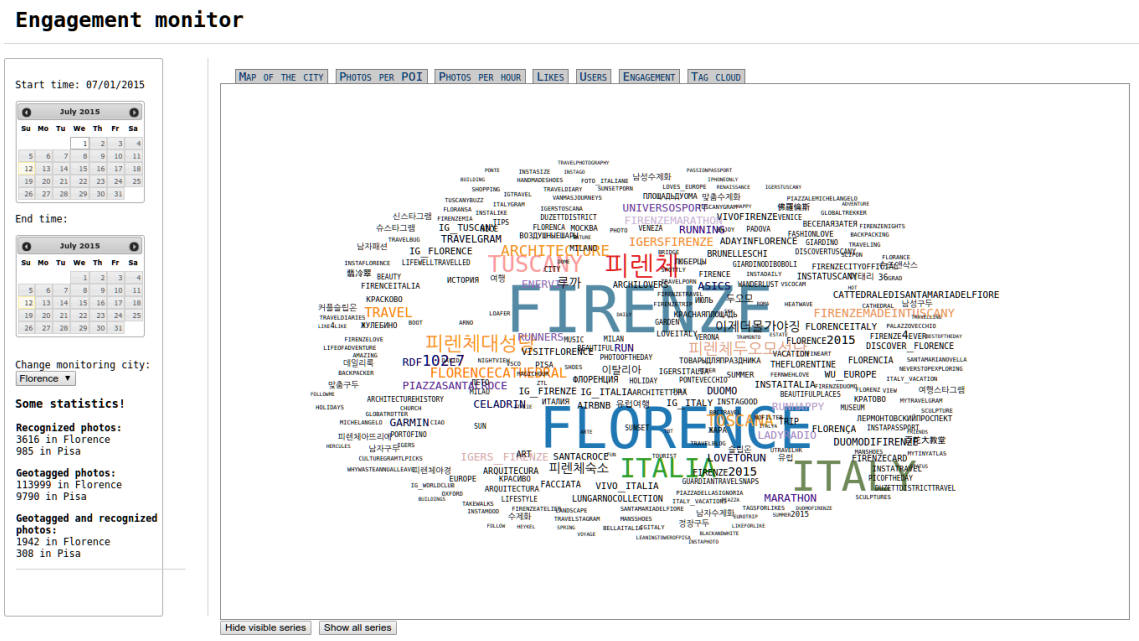


FIGURE 6.7: Tag cloud

# Chapter 7

# Tests results

As said before, two italian cities have been monitored by our prototype: Florence and Pisa.

## 7.1 Florence

### 7.1.1 Available knowledge base

- **1526** tags in different languages
- **147** points of interest
- **895** total photos

### 7.1.2 Crawler volumes

- **290000** c.a. crawled photos collected in 3 months
- **3200** crawled photos per day on average (more on weekends)
- **130** photos considered reliably recognized per day on average

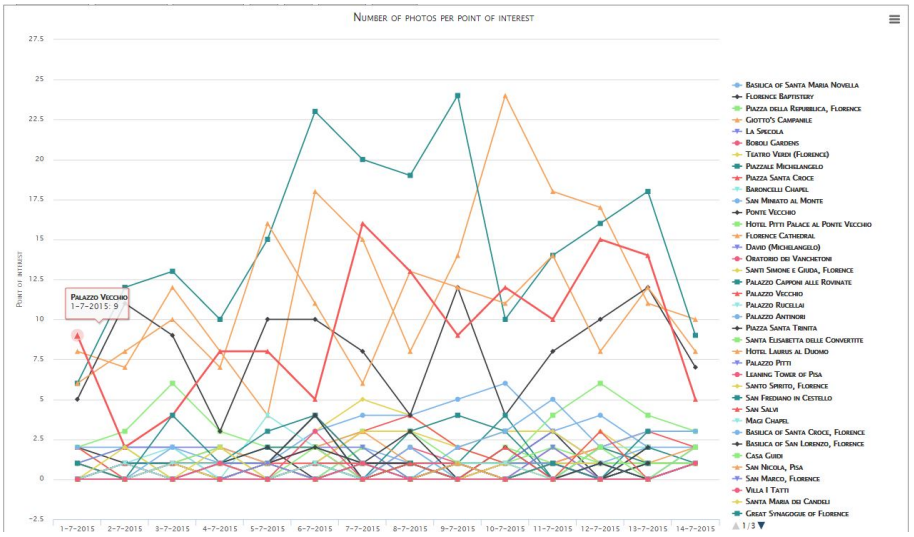


FIGURE 7.1: Points of interest in Florence

### 7.1.3 Results

The most engaging point of interests seem to be the classical visitor attractions of the city: Ponte Vecchio, the area of Piazza del Duomo (Florence Cathedral, Giotto's Campanilie), Piazzale Michelangelo and the Basilica of Santa Maria Novella.

## 7.2 Pisa

### 7.2.1 Available knowledge base

- **100** tags
- **43** points of interest
- **286** photos

### 7.2.2 Crawler volumes

- **34000** c.a. crawled photos collected in one month
- **850** crawled photos per day on average (more on weekends)
- **60** photos considered reliably recognized per day on average

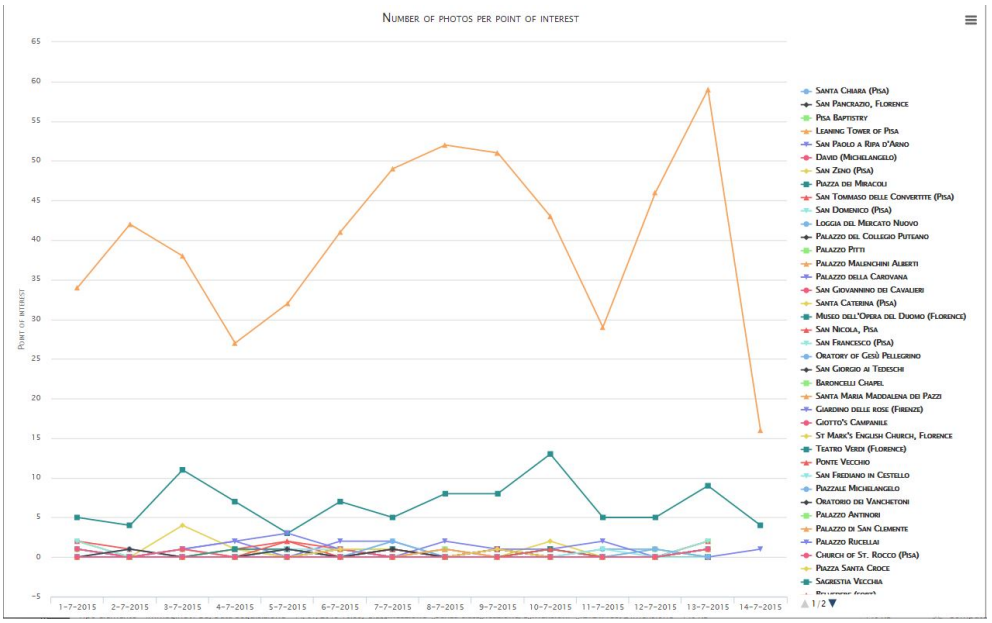


Figure 7.2: Points of interest in Pisa

### 7.2.3 Results

Here we have an extremely high concentration of interest for the Leaning Tower and Piazza dei Miracoli, as expected.

## 7.3 General performances

We had at our disposal a machine with 4 GB RAM and 4 cores; with this technology, our recognizer can successfully process 500 images every 4 minutes and 30 seconds.
The recognition process itself did not perform very well, partly because of:

- the very high volume of irrelevant photos (selfies, food photos, shopping photos...)
- the similarities between buildings (especially in historical cities like Florence and Pisa) that may lead the recognizer to mistakes
- the use of filters that may alter the photos in a way that the recognizer cannot process them well.

In order to perform a correct photo classification, we had to choose a **threshold** for the **confidence** parameter of the recognizer output.
On a sample of randomly chosen **100 photos**, we varied the confidence threshold and by manual inspection we obtained the following confusion matrices:

**Recognizer outcome, th=0.10**

| actual value | **p** | **n** | |
|---|---|---|---|
| **p′** | 21 | 5 | P′ |
| **n′** | 62 | 12 | N′ |
| **total** | P | N | |

**Recognizer outcome, th=0.15**

| actual value | **p** | **n** | |
|---|---|---|---|
| **p′** | 21 | 9 | P′ |
| **n′** | 6 | 53 | N′ |
| **total** | P | N | |

| Threshold | Precision | Recall | Sensitivity | Accuracy | F-measure |
|---|---|---|---|---|---|
| **0.10** | 25% | 80% | 16% | 33% | 28% |
| **0.15** | 77% | 70% | 89% | 74% | 73% |
| **0.20** | 76% | 22% | 94% | 63% | 44% |

TABLE 7.1: Precision, recall, sensitivity and accuracy of the system at different thresholds

**Recognizer outcome, th=0.20**

| actual value | **p** | **n** | |
|---|---|---|---|
| **p′** | 10 | 34 | P′ |
| **n′** | 3 | 53 | N′ |
| **total** | P | N | |

Observing this results, we can make the following considerations:

- below 0.1 there were too many false positives to consider a threshold below 0.1;
- with values between 0.1 and 0.15 the precision of the recognizer rises dramatically; at 0.15 the number of false positives is very low
- recall at 0.10 and 0.15 is more or less the same, while precision and accuracy are much better
- with values over 0.20 the recognizer has 100% precision, but very low recall (lots of correctly recognized photos are discarded).
- according to the F-measure, the threshold that balances precision and recall is 0.15 (that is also the threshold with higher accuracy).

Basing our decision on these considerations, we chose 0.15 as threshold, as trade-off between precision and recall. As said before, in this sample we noticed that

photos with common Instagram filters were not recognized, even when there was a point of interest in plain sight.

In Figure 7.3 and Figure 7.4 we report some graphs regarding the Florence crawling volumes and the percentage of recognized posts. Data before 4th July, 2015 is relative to posts associated only to 'florence' tags (e.g. florence2015, florencebynight etc.).

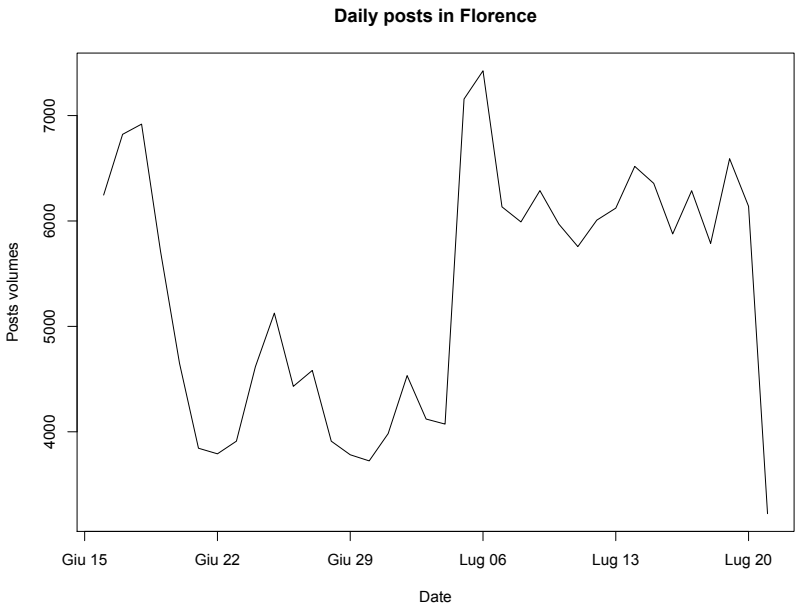


FIGURE 7.3: Graph of photos retrieved per day in Florence

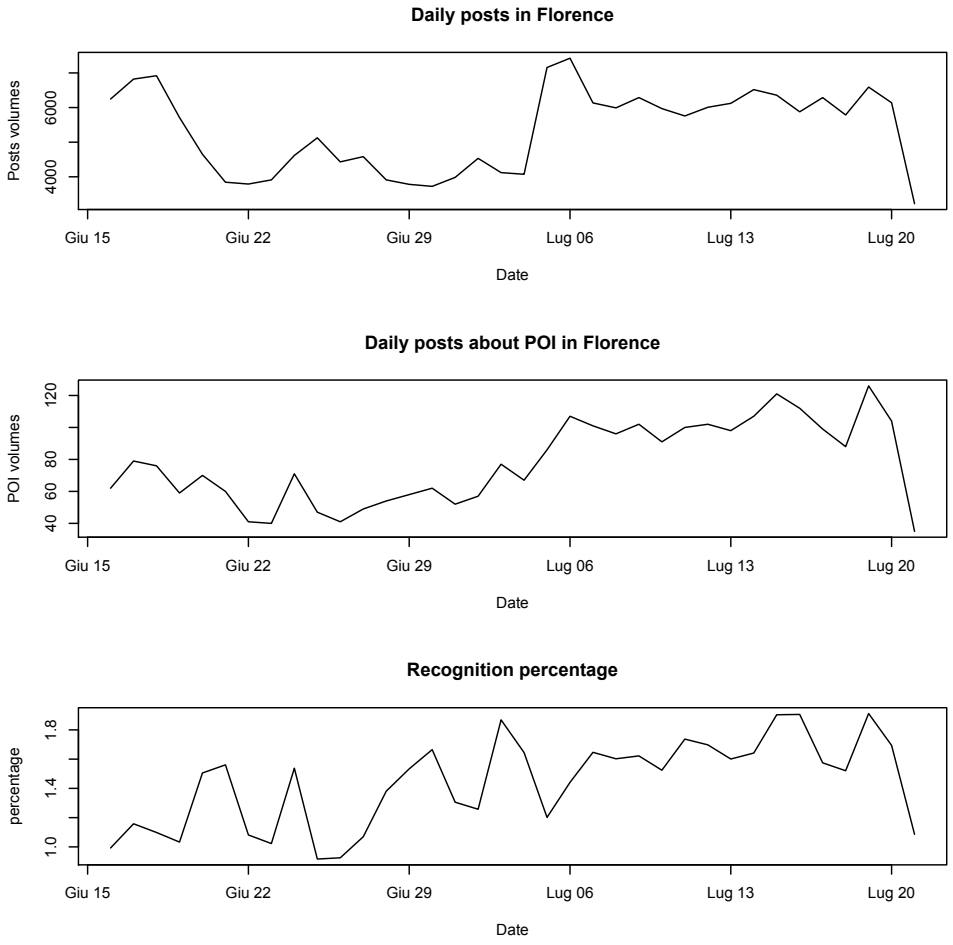


FIGURE 7.4: Other graphs regarding recognition performances

We can see that recognition percentage is mainly constant and its trend follows more or less the volume of retrieved photos: recognized photos are always between 1% and 2% of total photos.

### 7.3.1 Geotagging VS tag based retrieval

Once enough data has been collected, we can compare its performances against information retrieved with geotagging.

We extracted two sets of post taken in 14th July, 2015 in the city of Florence, one set of photos retrieved separately with geolocalization and the other one of photos retrieved with our method in the same period. Here's what transpired:

- the tag based approach retrieved three times more photos than geotagged approach (6528 vs 2741);
- 371 photos were retrieved from both approaches;
- 107 photos have been recognized from the tag based set, against the 28 of the geotagged set;
- only 5 photos were recognized and retrieved from both approaches.

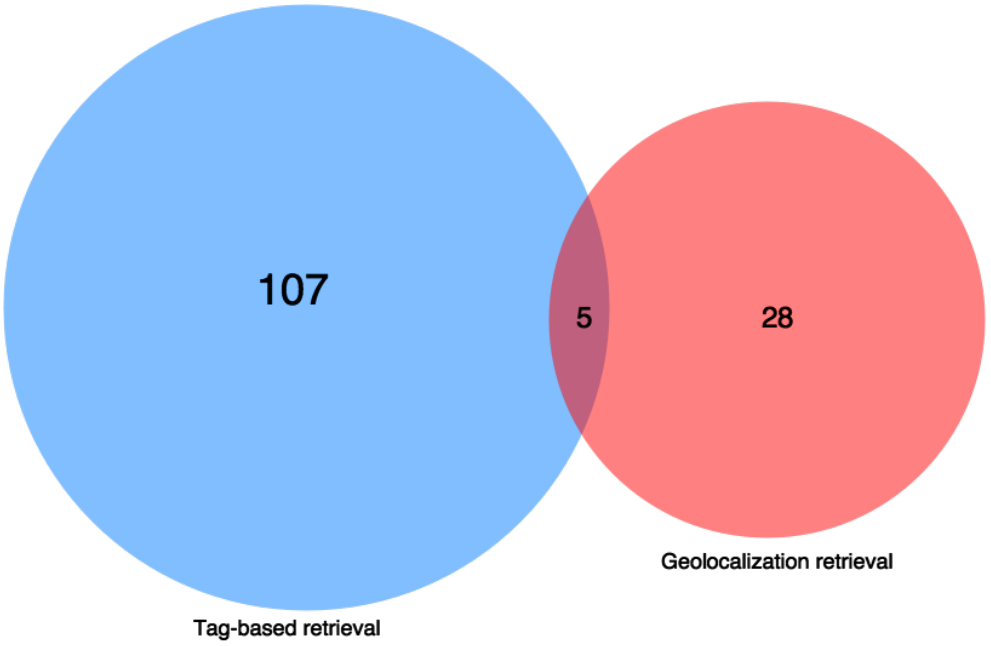


FIGURE 7.5: A Venn diagram representing the two approaches and their intersection

The tag based approach seems to collect more data than the geotagged one, and (most important) more data *relevant* to our purposes. The intersection of the two sets of photos is, however, very small (we have only 371 photos both geotagged and tagged, and only 5 of them were recognized); a definitive preference of an approach over the other one may cause a significant data missing.

### 7.3.2 Data relevance for different hours of the day

The time of the day in which we can obtain more recognized photos seems to be between 2.00 PM and 7.00 PM. This could mean that the majority of tourists

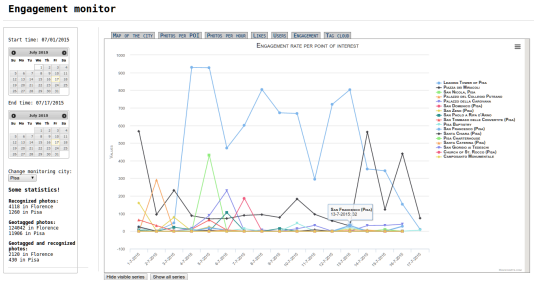


FIGURE 7.6: Example of Engagement chart between 1st and 17th July, 2015

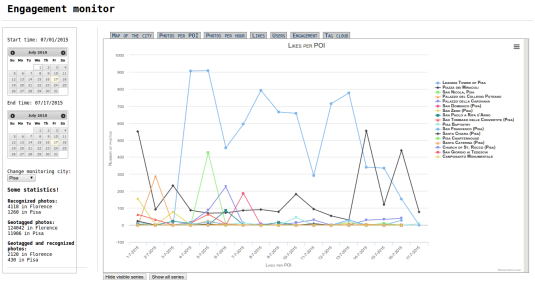


FIGURE 7.7: Example of Likes chart between 1st and 17th July, 2015

dedicate these hours of the day to visit a particular city, since the photo taken in this period have more engagement significance; but it could also indicate that people take lots of photos in the afternoon and for this reason we notice an increase of recognized photos. In figure 7.8 an example is presented, in the space of a week.

### 7.3.3 Likes,comments and engagement

We noticed that the *Likes* chart and the *Engagement* chart present more or less the same behavior (Figure 7.6 and 7.7): this means that normally for each post the number of comments is much smaller that number of likes. This is a normal behavior in social media, since normally a comment (of whatever nature, enthusiastic or deprecatory) represents higher participation and interest to the post content. Such considerations can lead to give more weight in our engagement formula to comments than likes.

### 7.3.4 Instagram rate limits

The rate limit of 5,000 requests per hour (10,000 in our case, since we use 2 client-ids) represents the most important bottleneck of the system: we still manage to retrieve the photos, but the systems exceeds the hourly limit frequently and it introduces a latency of some hours in our result set. This is caused mainly by the large set of tags to crawl, that may result in a blocking in case of a sudden traffic burst (especially in afternoon hours, as we saw in the previous section).

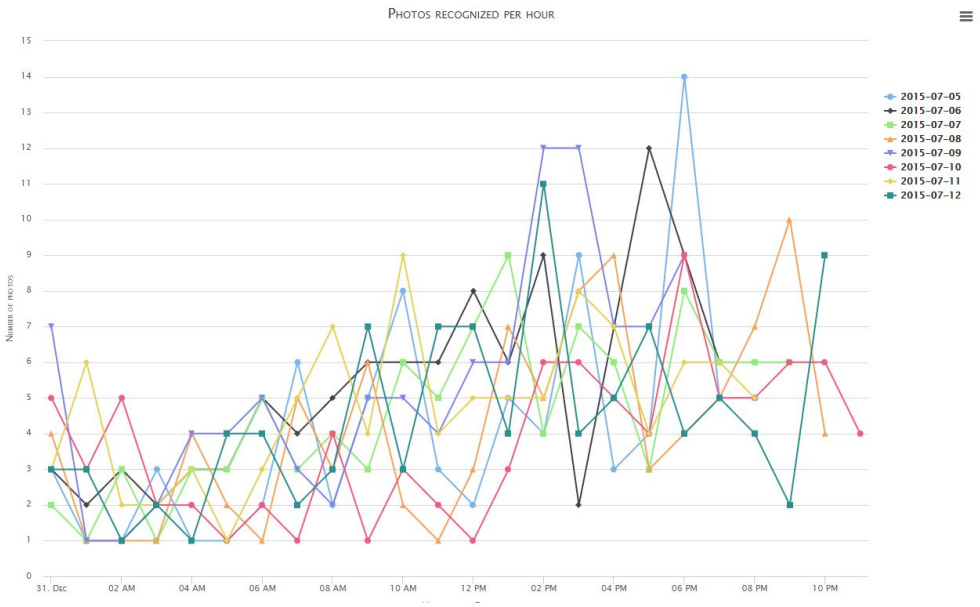


Figure 7.8: Relevant photos during the day between 5th and 12th July, 2015

### 7.3.5 User contribution

In the Users tab, we can observe (Figure 7.9 that most users take few relevant photos every day, and only for that day, in general without making any contribution in the following days.
This graph shows also the number of distinct visitors during the chosen time period.

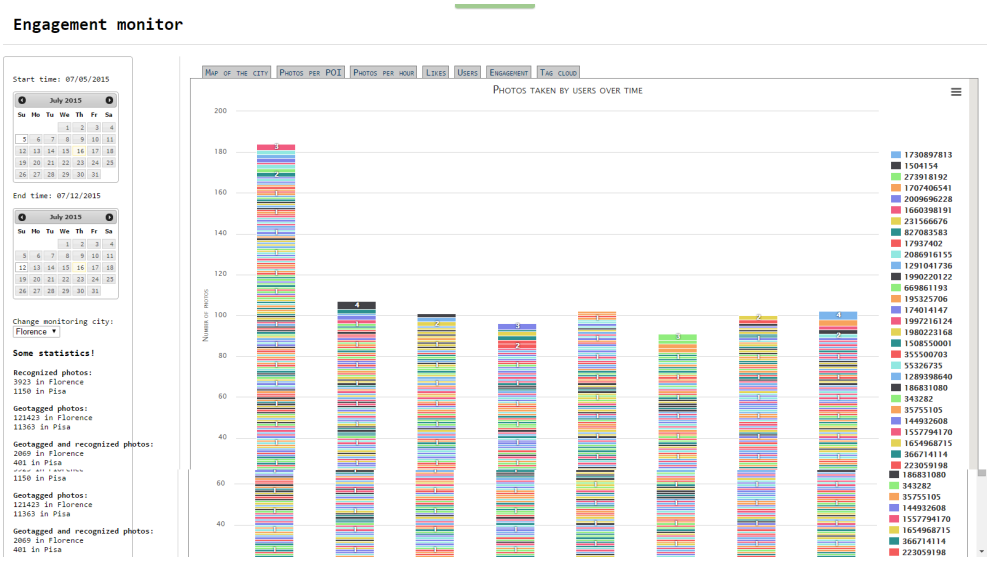


Figure 7.9: Contributions of the users, between 5th and 12th July, 2015

### 7.3.6 Tag cloud analysis

The study of the tag cloud gives us useful information about the type of tourists and events in the city, like:

- the language of the most popular tags may indicate an abundant participation of tourists of particular countries
- certain keywords may indicate the minor or major participation to events. A significant example is presented in figure 7.10: the Florence tag cloud between 4th and 5th July, 2015 presents a set of Korean words that are use even more frequently than #florence or #firenze, possibly meaning a great afflux of people of Korean visitors in those two days.

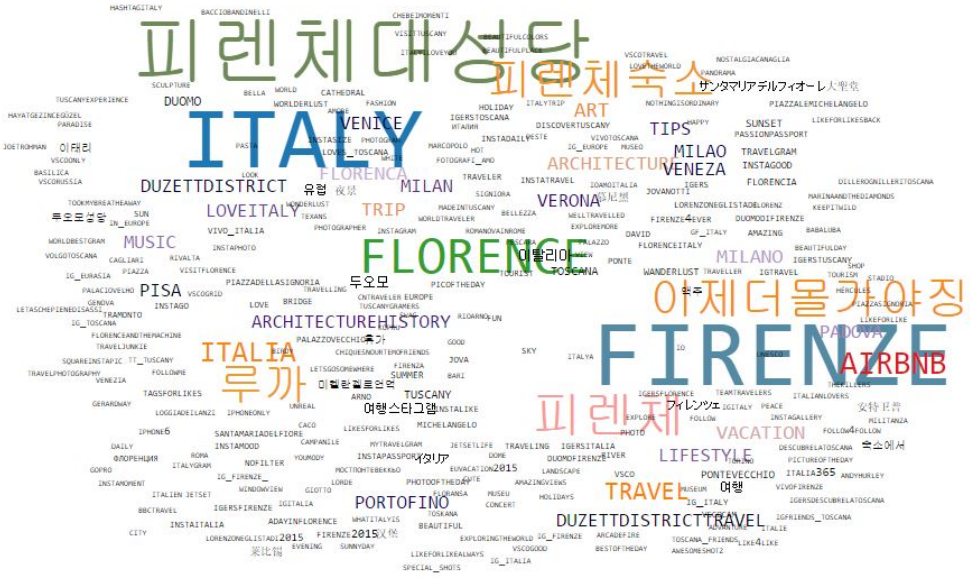


FIGURE 7.10: Florence tag cloud between 4th and 5th July, 2015

# Chapter 8

# Conclusions

In our work, a system for monitoring touristic interest in a city has been presented; it is based on **Instagram** photos and on an image **recognizer** trained with information taken automatically from reliable sources on the Internet.
The overall system is divided in three parts:

- a **knowledge base** building system in which we build the knowledge for the training process of the recognizer, that consist of a list of **points of interest** of a city, a set of photos associated with each point of interest and a list of **translations** of the name of the city
- a **crawling** system that retrieves **photos** associated with certain keywords about the city and **engagement data** associated with those photos
- the **recognizer** itself, to measure the relevance of collected photos on criteria based on their actual presence of points of interest in them.

A **web interface** was developed too, to visualize easily the performances of our system and the statistics that may be extrapolated from the collected data.

Both the knowledge base building process and the photo retrieval process were done not with a standard geolocation approach, but with a purely **tag-based approach**, and various comparisons were made between the two approaches. Our tests show that the tag-based approach gives back more results than the geolocation one, and more importantly we can obtain twice as many relevant results.

However, preferring one system over another may cause data missing, because the intersection between the results of the two methods is very small.

Most of the subjects of the photos are not really relevant to our purposes (selfies, food photos, shopping products...), and the recognizer has its flaws; but we still can obtain plenty of information regarding the behaviors of the tourists in a particular city, not only limited to their appreciation of the various points of interests in the city, but also about their nationality, movements and participation to events.

# Chapter 9

# Future work

Our system is designed to be as modular as possible, so that future developers can add or improve parts of it without modifying its general structure. Also, the data collected so far from our system can be analyzed (together with future data collected) to extract useful information.

## 9.1 System improvements

The following system improvements may be done:

- **Local DBPedia(s) analysis**: as said before, the DBPedias of different countries present different structures and ontologies. We can't extend our crawling process, based on the English DBPedia, to other DBPedias, even though surely local DBpedias hosts more complete information about their cities. A customized crawling for some DBpedias of languages different than English could be studied and eventually implemented.
- **Different POI sources**: DBpedia has been used as our source for building the list of points of interest; but since its constantly changing nature may be a problem for information retrieval in the future, it would be useful a study of more stable sources.

- **Different image sources**: Wikimedia Commons and Instagram are only two of many image sources present on World Wide Web; it could be useful a study of different image sources from which we can extract reliable information (Panoramio, Flickr) and/or of a different source for crawling.
- **face recognizing**: a face recognizer module can be inserted into the system, to detect a significant presence of faces from photos obtained from the monitor, and possibly remove them to avoid the inevitable noise they produce for the recognizer, allowing the tool to process a clean photo.
- **tag relevance**: a tag may or may not be related to the monitored city, despite having the city name in it; we may use the recognizer to count the number of relevant photos annotated with a certain tag and eventually decide to stop the monitoring of that particular tag, if it doesn't retrieve enough photos with relevant content for a while.
- **Instagram rate limits**[26]: For a single client ID, only a limited number of requests is allowed every hour. Some optimization and workarounds may be done to increase this limit and the collectable amount of data, thus eliminating the performance bottleneck.
- **dynamic knowledge base**: photos with a particularly high value of confidence may be added dynamically to the pre-existent knowledge base during the crawling process.

## 9.2 Data analysis

Many types of analysis[27] are possible on the data collected, here we propose some possible types:

- **Anomaly detection**: our dataset can be processed by an *outlier detection tool* to detect anomalies in storic series: e.g. an abnormal peak of photos of a certain POI during a specific day, allowing a possible detection of an out-of-ordinary event in that particular point of interest.
- **Movement analysis**: we can use collected data to analyze tourists' movement and therefore obtain the most popular touristic trip in a city, like in works as TourPedia[28] and TripBuilder[18].

- **Tag type analysis**: In addition to POI monitoring, an analysis of the most popular tags in a period of time may be done, to explore the popularity of certain tags related to particular events and/or in different languages, to study the tourist flow from different countries.

# Appendix A

# Code

In the following pages snippets of significant code are presented.

## A.1 Knowledge base creation

### A.1.1 DBPedia crawling and POI extraction

```
1 /**
2   function that retrieves a list of points of interest from
    DBPedia
3 **/
4
5 function extract_poi($mysql_connection, $city){
6 /* retrieval of needed string from an .ini file */
7                 $ini = parse_ini_file("/var/www/poi/config/dbpedia
    .ini");
8                 $db = sparql_connect($ini['sparql_url']);
9                 $prefixes = $ini['prefixes'];
10                return explore_subcat("category:
    Visitor_attractions_in_".$city, $prefixes, $mysql_connection,
    $city);
11                }
12
13
14 function explore_subcat($category, $prefixes, $mysql_connection,
    $city){
15 /* retrieval of the pages under the given category */
```

```
16                $query_string = "SELECT DISTINCT ?name ?image ?lat
     ?long ?wikipage
17                where {
18
19                        ?s dc:subject ".$category."  .
20                        ?s rdfs:label ?name .
21                        ?s foaf:depiction ?image .
22                        ?s geo:lat ?lat .
23                        ?s geo:long ?long .
24                        ?s foaf:isPrimaryTopicOf ?wikipage
25                        FILTER (langMatches(lang(?name), 'en'))
26                        }";                          ;
27                $pages = $this->execute_query($query_string,
    $prefixes);
28                if ($pages != 0){
29                        foreach($pages as $page) {
30                          echo $page;
31                                $this->insert_in_db($page,
    $mysql_connection, $city);
32                                }
33                        }
34 /* crawling of the subcategories */
35                $query_string = "SELECT DISTINCT ?subject
36                where {
37
38                {
39                        ?subject skos:broader ".$category.".
40                }
41                }";
42                $result2 = $this->execute_query($query_string,
    $prefixes);
43                if ($result2 != 0){
44                foreach ($result2 as $row)
45                        foreach ($row as $r){
46                        /* recursion on the given subcategory, if
    it exists */
47                                $this->explore_subcat("<".$r.">",
    $prefixes, $mysql_connection, $city);
48                        }
49                }
50                else return;
51                /* the algorithm stops when no more subcategories
    are available for the given category */
52         }
```

## A.2 Instagram crawler

### A.2.1 Post retrieval

```
1
2 $ini = parse_ini_file("/var/www/poi/config/instagram.ini");
3   $connection = connect_to_db();
4   mysql_select_db('poi_db');
5
6   $tags = perform_query($connection, "SELECT name, city_name from
     tags order by name");
7   $client = $ini['client1'];
8   $client_1 = $ini['client'];
9   if ($tags){
10  while ($row = mysql_fetch_row($tags))
11    {
12      $tag = $row[0];
13      $city = $row[1];
14      $url = "https://api.instagram.com/v1/tags/".$tag."/media/
     recent?count=33&client_id=".$client;
15      echo $url;
16      get_results($url, $tag, $city);
17      $temp = $client;
18      $client = $client_1;
19      $client_1 = $temp;
20    }
21  }
22
23  function my_sort($a, $b)
24 {
25    if ($a['created_time'] > $b['created_time']) {
26        return 1;
27    } else if ($a['created_time'] < $b['created_time']) {
28        return -    1;
29    } else {
30        return 0;
31    }
32 }
33
```

```
34  function get_results($url, $current_tag, $city){
35
36    $conn=connect_to_db();
37    mysql_select_db('poi_db');
38    $session_start_time = strval(time());
39
40    $got_all = false;
41    echo $url;
42    $n_pics = 0; $n_calls = 0; $n_loc = 0; $n_notloc = 0;
43     $sql = "SELECT creation_time from crawler where tag = '".
    $current_tag."' order by creation_time desc limit 1";
44     $time = perform_query($conn, $sql);
45      if (mysql_num_rows($time) == 0)
46        $last_time = round(microtime(true)) - 432000; /* dump of
    data of last week */
47      else
48      {
49      $last_time = mysql_fetch_array($time)[0];
50      echo $last_time;
51      $last_time = strtotime($last_time);
52      }
53     while(!$got_all) {
54       $result = get_pics($url);
55       /* here we sort response by date */
56       usort($result['data'], 'my_sort');
57       $session = 0;
58       foreach($result['data'] as &$pic)
59       {
60         if ($pic['created_time'] > $last_time)
61         {
62           $sql = "INSERT INTO crawler (user, media_id, media_url,
    creation_time, tag, city_name)
63             VALUES ('{$pic['user']['id']}', '{$pic['id']}', '{$pic
    ['images']['standard_resolution']['url']}', FROM_UNIXTIME('{
    $pic['created_time']}*1000'), '$current_tag', '$city')
64           ");
65           perform_query($mysql_connection, $sql);
66           foreach ($pic['tags'] as $tag){
67           $sql = "
68             INSERT INTO media_tags (media_id, tag)
69             VALUES ('".$pic['id']."', '".$tag."')
70           ";
71           perform_query($mysql_connection, $sql);
72           }
```

```
73
74
75
76          $n_pics++;
77          }
78          else
79            $session++;
80          /* old photo found; if at least 33 old photos are found
     the crawling stops */
81        }
82
83        if (!(isset($result['pagination']['next_url'])) || $session
     >= 33) {
84          $got_all = true;
85        } else {
86          $url = $result['pagination']['next_url'];
87        }
88      }
89
90
91    }
92
93    function get_pics($url) {
94      $ch = curl_init();
95      curl_setopt($ch, CURLOPT_URL, $url);
96      curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
97      curl_setopt($ch, CURLOPT_TIMEOUT, 20);
98      $result = curl_exec($ch);
99      curl_close($ch);
100     $result = json_decode($result, true);
101     return $result;
102   }
```

### A.2.2 Content of crontab

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
```

```
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

*/20 * * * * nohup php /var/www/poi/crawler/crawler.php >/dev/null 2>&1 &
*/10 * * * * php /var/www/poi/crawler/multi_recognizer.php >/dev/null 2>&1 &
0 */1 * * * nohup php /var/www/poi/engagement/likes.php >/dev/null 2>&1 &
*/30 * * * * nohup php /var/www/poi/engagement/followers.php >/dev/null 2>&1 &
```

## A.3 Dump of the DB structure

```
-- phpMyAdmin SQL Dump
-- version 4.0.10deb1
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: Jul 12, 2015 at 04:50 PM
-- Server version: 5.6.19-0ubuntu0.14.04.1
-- PHP Version: 5.5.9-1ubuntu4.9

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Database: `poi_db`
--
CREATE DATABASE IF NOT EXISTS `poi_db`;
USE `poi_db`;

-- --------------------------------------------------------

--
-- Table structure for table `cities`
--

CREATE TABLE IF NOT EXISTS `cities` (
  `city_id` int(11) NOT NULL,
  `city_name` varchar(128) NOT NULL,
  PRIMARY KEY (`city_id`),
```

```
  UNIQUE KEY `city_name` (`city_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `crawler`
--

CREATE TABLE IF NOT EXISTS `crawler` (
  `media_id` varchar(255) NOT NULL,
  `media_url` varchar(255) NOT NULL,
  `tag` varchar(128) NOT NULL,
  `label` varchar(128) NOT NULL,
  `confidence` double NOT NULL,
  `user` varchar(128) NOT NULL,
  `city_name` varchar(128) NOT NULL,
  `creation_time` datetime NOT NULL,
  PRIMARY KEY (`media_id`),
  UNIQUE KEY `media_id_2` (`media_id`),
  KEY `creation_time` (`creation_time`),
  KEY `user` (`user`),
  KEY `label` (`label`),
  KEY `user_2` (`user`),
  KEY `label_2` (`label`),
  KEY `user_3` (`user`),
  KEY `label_3` (`label`),
  KEY `creation_time_2` (`creation_time`),
  KEY `city_name` (`city_name`),
  KEY `label_4` (`label`),
  KEY `user_4` (`user`),
  KEY `creation_time_3` (`creation_time`),
  FULLTEXT KEY `media_id` (`media_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `followers`
--

CREATE TABLE IF NOT EXISTS `followers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user` varchar(128) NOT NULL,
  `n_followers` varchar(128) NOT NULL,
  PRIMARY KEY (`user`),
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=129635 ;

-- --------------------------------------------------------

--
-- Table structure for table `geoloc`
--
```

```
CREATE TABLE IF NOT EXISTS `geoloc` (
  `media_id` varchar(255) CHARACTER SET utf8 NOT NULL,
  `lat` float NOT NULL,
  `lon` float NOT NULL,
  PRIMARY KEY (`media_id`),
  KEY `media_id` (`media_id`),
  KEY `media_id_2` (`media_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


-- --------------------------------------------------------

--
-- Table structure for table `images`
--

CREATE TABLE IF NOT EXISTS `images` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(50) DEFAULT NULL,
  `image` varchar(255) DEFAULT NULL,
  `city_name` varchar(128) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=18690 ;

-- --------------------------------------------------------

--
-- Table structure for table `likes`
--

CREATE TABLE IF NOT EXISTS `likes` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `media_id` varchar(255) NOT NULL,
  `n_likes` int(11) NOT NULL,
  `n_comments` int(11) NOT NULL,
  PRIMARY KEY (`media_id`),
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=323628 ;

-- --------------------------------------------------------

--
-- Table structure for table `media_tags`
--

CREATE TABLE IF NOT EXISTS `media_tags` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `media_id` varchar(255) NOT NULL,
  `tag` varchar(128) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `media_id_2` (`media_id`),
  KEY `tag_2` (`tag`),
  KEY `tag_3` (`tag`),
  FULLTEXT KEY `media_id` (`media_id`),
  FULLTEXT KEY `tag` (`tag`)
```

```
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=9753436 ;

-- --------------------------------------------------------

--
-- Table structure for table `poi`
--

CREATE TABLE IF NOT EXISTS `poi` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(50) DEFAULT NULL,
  `lat` varchar(8) DEFAULT NULL,
  `lon` varchar(8) DEFAULT NULL,
  `wikipage` varchar(100) DEFAULT NULL,
  `city_name` varchar(128) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=11755 ;

-- --------------------------------------------------------

--
-- Table structure for table `tags`
--

CREATE TABLE IF NOT EXISTS `tags` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(50) CHARACTER SET utf8 DEFAULT NULL,
  `date` date DEFAULT NULL,
  `city_name` varchar(128) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=8283 ;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

## A.4 JSON responses

### A.4.1 MediaWiki: Knowledge base images retrieval

```
{
    "warnings": {
        "query": {
            "*": "Formatting of continuation data has changed. To
    receive raw query-continue data, use the 'rawcontinue'
    parameter. To silence this warning, pass an empty string for '
    continue' in the initial query."
        }
```

```
    },
    "batchcomplete": "",
    "continue": {
        "gcmcontinue": "file|464c4f52454e4345204954414c5920504f4e5
    4452d5645434348494f2d30312e4a5047|39191347",
        "continue": "gcmcontinue||"
    },
    "query": {
        "pages": {
            "39342379": {
                "pageid": 39342379,
                "ns": 6,
                "title": "File:03 2015 Ponte Vecchio-Arno-Portico-
    ordine tuscanico, arco a tutto sesto (Firenze) Photo Paolo
    Villa FOTO9247.JPG",
                "imagerepository": "local",
                "imageinfo": [
                    {
                        "url": "https://upload.wikimedia.org/
    wikipedia/commons/3/39/03_2015_Ponte_Vecchio-Arno-Portico-
    ordine_tuscanico%2C_arco_a_tutto_sesto_%28Firenze%29
    _Photo_Paolo_Villa_FOTO9247.JPG",
                        "descriptionurl": "https://commons.
    wikimedia.org/wiki/File:03_2015_Ponte_Vecchio-Arno-Portico-
    ordine_tuscanico,_arco_a_tutto_sesto_(Firenze)
    _Photo_Paolo_Villa_FOTO9247.JPG"
                    }
                ]
            },
            "39342376": {
                "pageid": 39342376,
                "ns": 6,
                "title": "File:03 2015 Ponte Vecchio-Arno-Portico-
    ordine tuscanico, arco a tutto sesto (Firenze) Photo Paolo
    Villa FOTO9247bis.JPG",
                "imagerepository": "local",
                "imageinfo": [
                    {
                        "url": "https://upload.wikimedia.org/
    wikipedia/commons/4/42/03_2015_Ponte_Vecchio-Arno-Portico-
    ordine_tuscanico%2C_arco_a_tutto_sesto_%28Firenze%29
    _Photo_Paolo_Villa_FOTO9247bis.JPG",
```

```
34                      "descriptionurl": "https://commons.
    wikimedia.org/wiki/File:03_2015_Ponte_Vecchio-Arno-Portico-
    ordine_tuscanico,_arco_a_tutto_sesto_(Firenze)
    _Photo_Paolo_Villa_FOTO9247bis.JPG"
35                  }
36              ]
37          },
38          "39344050": {
39              "pageid": 39344050,
40              "ns": 6,
41              "title": "File:03 2015 Ponte Vecchio-Arno-Portico-
    ordine tuscanico, arco a tutto sesto (Firenze) Photo Paolo
    Villa FOTO9250.JPG",
42              "imagerepository": "local",
43              "imageinfo": [
44                  {
45                      "url": "https://upload.wikimedia.org/
    wikipedia/commons/a/a8/03_2015_Ponte_Vecchio-Arno-Portico-
    ordine_tuscanico%2C_arco_a_tutto_sesto_%28Firenze%29
    _Photo_Paolo_Villa_FOTO9250.JPG",
46                      "descriptionurl": "https://commons.
    wikimedia.org/wiki/File:03_2015_Ponte_Vecchio-Arno-Portico-
    ordine_tuscanico,_arco_a_tutto_sesto_(Firenze)
    _Photo_Paolo_Villa_FOTO9250.JPG"
47                  }
48              ]
49          },
50          "39344041": {
51              "pageid": 39344041,
52              "ns": 6,
53              "title": "File:03 2015 Ponte Vecchio-Arno-Portico-
    ordine tuscanico, arco a tutto sesto (Firenze) Photo Paolo
    Villa FOTO9257.JPG",
54              "imagerepository": "local",
55              "imageinfo": [
56                  {
57                      "url": "https://upload.wikimedia.org/
    wikipedia/commons/0/0b/03_2015_Ponte_Vecchio-Arno-Portico-
    ordine_tuscanico%2C_arco_a_tutto_sesto_%28Firenze%29
    _Photo_Paolo_Villa_FOTO9257.JPG",
58                      "descriptionurl": "https://commons.
    wikimedia.org/wiki/File:03_2015_Ponte_Vecchio-Arno-Portico-
    ordine_tuscanico,_arco_a_tutto_sesto_(Firenze)
    _Photo_Paolo_Villa_FOTO9257.JPG"
```

```
                    }
                ]
            },
...
```

### A.4.2 Instagram: tag retrieval

```
{
    "meta": {
        "code": 200
    },
    "data": [
        {
            "media_count": 1946821,
            "name": "firenze"
        },
        {
            "media_count": 19472,
            "name": "firenze4ever"
        },
        {
            "media_count": 10958,
            "name": "firenzemadeintuscany"
        },
        {
            "media_count": 10299,
            "name": "firenzecard"
        },
        {
            "media_count": 3938,
            "name": "firenze2014"
        },
        {
            "media_count": 3801,
            "name": "firenze2015"
        },
        {
            "media_count": 3162,
            "name": "firenzebynight"
        },
        {
            "media_count": 2212,
```

```
            "name": "firenze2013"
        },
        {
            "media_count": 1878,
            "name": "firenzemarathon"
        },
        {
            "media_count": 1688,
            "name": "firenzegram"
        },
        {
            "media_count": 1307,
            "name": "firenzeforever"
        },
        {
            "media_count": 1256,
            "name": "firenzetoday"
        },
        {
            "media_count": 1147,
            "name": "firenzenelcuore"
        },
        {
            "media_count": 1092,
            "name": "firenzeitalia"
        },
...
```

### A.4.3 Instagram: post retrieval

```
{
    "pagination": {
        "next_max_tag_id": "1024607137612482170",
        "deprecation_warning": "next_max_id and min_id are
    deprecated for this endpoint; use min_tag_id and max_tag_id
    instead",
        "next_max_id": "1024607137612482170",
        "next_min_id": "1024613840405115709",
        "min_tag_id": "1024613840405115709",
        "next_url": "https://api.instagram.com/v1/tags/florence/
    media/recent?count=33&client_id=83afcd79589440e0a4f5e8eba16efe7
    9&max_tag_id=1024607137612482170"
```

```
    },
    "meta": {
        "code": 200
    },
    "data": [
        {
            "attribution": null,
            "tags": [
                "tuscany",
                "panorama",
                "cityhopping",
                "firenze",
                "duomo",
                "italy",
                "florence",
                "view"
            ],
            "location": {
                "latitude": 43.762958184,
                "name": "Forte di Belvedere",
                "longitude": 11.256113019,
                "id": 289038127
            },
            "comments": {
                "count": 0,
                "data": []
            },
            "filter": "Normal",
            "created_time": "1436363509",
            "link": "https://instagram.com/p/44KNv1S7M9/",
            "likes": {
                "count": 0,
                "data": []
            },
            "images": {
                "low_resolution": {
                    "url": "https://scontent.cdninstagram.com/
hphotos-xfa1/t51.2885-15/s320x320/e15/11356793_1616653911905886
_2110550807_n.jpg",
                    "width": 320,
                    "height": 320
                },
                "thumbnail": {
```

```
50                      "url": "https://scontent.cdninstagram.com/
    hphotos-xfa1/t51.2885-15/s150x150/e15/11356793_1616653911905886
    _2110550807_n.jpg",
51                      "width": 150,
52                      "height": 150
53                  },
54                  "standard_resolution": {
55                      "url": "https://scontent.cdninstagram.com/
    hphotos-xfa1/t51.2885-15/s640x640/e35/sh0.08/11356793_161665391
    1905886_2110550807_n.jpg",
56                      "width": 640,
57                      "height": 640
58                  }
59              },
60              "users_in_photo": [],
61              "caption": {
62                  "created_time": "1436363509",
63                  "text": "Breathtaking views over the city of
    Florence from the Forte di Belvedere earlier today  #panorama #
    view #firenze #florence #tuscany #italy #cityhopping #duomo",
64                  "from": {
65                      "username": "marievandeweyer_",
66                      "profile_picture": "https://instagramimages-a.
    akamaihd.net/profiles/profile_712964843_75sq_1385398269.jpg",
67                      "id": "712964843",
68                      "full_name": "Marie Vandeweyer"
69                  },
70                  "id": "1024613842720371374"
71              },
72              "type": "image",
73              "id": "1024613840405115709_712964843",
74              "user": {
75                  "username": "marievandeweyer_",
76                  "profile_picture": "https://instagramimages-a.
    akamaihd.net/profiles/profile_712964843_75sq_1385398269.jpg",
77                  "id": "712964843",
78                  "full_name": "Marie Vandeweyer"
79              }
80          },
81
82  ...
```

### A.4.4 Instagram: media JSON


```
{
    "meta": {
        "code": 200
    },
    "data": {
        "attribution": null,
        "tags": [
            "arthistoryabroad",
            "florence"
        ],
        "location": {
            "latitude": 43.770858333,
            "longitude": 11.255121667
        },
        "comments": {
            "count": 0,
            "data": []
        },
        "filter": "Lo-fi",
        "created_time": "1436880647",
        "link": "https://instagram.com/p/5Hkk6xw3fg/",
        "likes": {
            "count": 1,
            "data": [
                {
                    "username": "carachamber123",
                    "profile_picture": "https://igcdn-photos-e-a.akamaihd.net/hphotos-ak-xaf1/t51.2885-19/s150x150/11356822_682674911876156_864718820_a.jpg",
                    "id": "1387869787",
                    "full_name": ""
                }
            ]
        },
        "images": {
            "low_resolution": {
                "url": "https://scontent.cdninstagram.com/hphotos-xfa1/t51.2885-15/s320x320/e15/11426684_493018397518764_1249716328_n.jpg",
                "width": 320,
                "height": 320
            },
```

```
            "thumbnail": {
                "url": "https://scontent.cdninstagram.com/hphotos-xfa1/t51.2885-15/s150x150/e15/11426684_493018397518764_1249716328_n.jpg",
                "width": 150,
                "height": 150
            },
            "standard_resolution": {
                "url": "https://scontent.cdninstagram.com/hphotos-xfa1/t51.2885-15/s640x640/e35/sh0.08/11426684_493018397518764_1249716328_n.jpg",
                "width": 640,
                "height": 640
            }
        },
        "users_in_photo": [],
        "caption": {
            "created_time": "1436880647",
            "text": "Last night in this beautiful city then onto Rome... #arthistoryabroad #florence",
            "from": {
                "username": "obott21",
                "profile_picture": "https://igcdn-photos-e-a.akamaihd.net/hphotos-ak-xaf1/t51.2885-19/s150x150/11253702_852502421501612_1486869834_a.jpg",
                "id": "193234316",
                "full_name": "Ollie Bott"
            },
            "id": "1028951908586583085"
        },
        "type": "image",
        "id": "1028951906564929504_193234316",
        "user": {
            "username": "obott21",
            "profile_picture": "https://igcdn-photos-e-a.akamaihd.net/hphotos-ak-xaf1/t51.2885-19/s150x150/11253702_852502421501612_1486869834_a.jpg",
            "id": "193234316",
            "full_name": "Ollie Bott"
        }
    }
}
```

### A.4.5 Instagram: user JSON

```
{
    "meta": {
        "code": 200
    },
    "data": {
        "username": "antoxmarino",
        "bio": "Follow me - all pictures are made by me - I live
    in Genoa Italy",
        "website": "",
        "profile_picture": "https://instagramimages-a.akamaihd.net
    /profiles/profile_1264687840_75sq_1397426966.jpg",
        "full_name": "Amamarino",
        "counts": {
            "media": 1185,
            "followed_by": 362,
            "follows": 322
        },
        "id": "1264687840"
    }
}
```

# Bibliography


[1] Maurizio Tesconi, Davide Gazzé, and Angelica Lo Duca. Socialtrends: a web application for monitoring and visualizing users in social media. In *Social Informatics*, pages 535–538. Springer Berlin Heidelberg, 2012.

[2] Graham Vickery and Sacha Wunsch-Vincent. *Participative web and user-created content: Web 2.0 wikis and social networking*. Organization for Economic Cooperation and Development (OECD), 2007.

[3] Daniel M Romero, Wojciech Galuba, Sitaram Asur, and Bernardo A Huberman. Influence and passivity in social media. In *Proc. European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 18–33. Springer Berlin Heidelberg, 2011.

[4] Robert Andrew Dunn. Identity theories and technology. In *Handbook of Research on Technoself*, volume 1, pages 26–44. IGI Global, 2012.

[5] Alessandro Lovari, Soojin KIM, Kelly Vibber, and Jeong-Nam Kim. Digitisation's impacts on publics: Public knowledge and civic conversation. *PRism*, 8(2), 2005.

[6] Rosemary Thackeray, Brad L Neiger, and Heidi Keller. Integrating social media and social marketing a four-step process. *Health promotion practice*, 13(2):165–168, 2012.

[7] Andreas M Kaplan and Michael Haenlein. Users of the world, unite! the challenges and opportunities of social media. *Business horizons*, 53(1):59–68, 2010.

[8] Rossella C Gambetti and Guendalina Graffigna. The concept of engagement a systematic analysis of the ongoing marketing debate. *International Journal of Market Research*, (52):801–826, 2010.

[9] Susan Murray. Digital images, photo-sharing, and our shifting notions of everyday aesthetics. *Journal of Visual Culture*, 7(2):147–163, 2008.

[10] Shane Ahern, Dean Eckles, Nathaniel S Good, Simon King, Mor Naaman, and Rahul Nair. Over-exposed?: privacy patterns and considerations in online and mobile photo sharing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 357–366. ACM, 2007.

[11] Oded Nov, M NAAMAN, and Ch Ye. Community photo sharing: Motivational and structural antecedents. *Retrieved March*, 15:2010, 2008.

[12] Oded Nov, Mor Naaman, and Chen Ye. Analysis of participation in an online photo-sharing community: A multidimensional perspective. *Journal of the American Society for Information Science and Technology*, 61(3):555–566, 2010.

[13] José Van Dijck. Flickr and the culture of connectivity: Sharing views, experiences, memories. *Memory Studies*, page 1750698010385215, 2010.

[14] Nadav Hochman and Lev Manovich. Zooming into an instagram city: Reading the local through social media. *First Monday*, 18(7), 2013.

[15] Stephanie Hays, Stephen John Page, and Dimitrios Buhalis. Social media as a destination marketing tool: its use by national tourism organisations. *Current Issues in Tourism*, 16(3):211–239, 2013.

[16] Thiago H Silva, Pedro OS de Melo, Jussara M Almeida, Juliana Salles, Antonio Loureiro, et al. A picture of instagram is worth more than a thousand words: Workload characterization and application. In *Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on*, pages 123–132. IEEE, 2013.

[17] Maurizio Tesconi, Francesco Ronzano, Andrea Marchetti, and Salvatore Minutoli. Semantify del. icio. us: Automatically turn your tags into senses. *The 7th International Semantic Web Conference*, page 67, 2008.

[18] Igo Ramalho Brilhante, Jose Antonio Macedo, Franco Maria Nardini, Raffaele Perego, and Chiara Renso. On planning sightseeing tours with tripbuilder. *Information Processing & Management*, 51(2):1–15, 2015.

[19] Frank Bergler. Application program interface, November 5 1996. US Patent 5,572,675.

[20] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *International Semantic Web Conference*, pages 722–735. Springer Berlin/Heidelberg, 2008.

[21] Markus Krötzsch, Denny Vrandečić, Max Völkel, Heiko Haller, and Rudi Studer. Semantic wikipedia. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(4):251–261, 2007.

[22] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

[23] M Dean and G Schreiber. Owl web ontology language reference. w3c recommandation. *Word Wide Web Consortium (W3C)*, 2004.

[24] Giuseppe Amato and Fabrizio Falchi. knn based image classification relying on local feature similarity. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, pages 101–108. ACM, 2010.

[25] Gina Schlesselman-Tarango. Searchable signatures: Context and the struggle for recognition. *Information Technology and Libraries*, 32(3):5–19, 2013.

[26] Christian Reuter and Simon Scholl. Technical limitations for designing applications for social media. In *Mensch & Computer*, pages 131–140, 2014.

[27] Lydia Manikonda, Yuheng Hu, and Subbarao Kambhampati. Analyzing user activities, demographics, social network structure and user-generated content on instagram. *arXiv preprint arXiv:1410.8099*, 2014.

[28] Stefano Cresci, Andrea D'Errico, Davide Gazzé, A Lo Duca, Andrea Marchetti, and Maurizio Tesconi. Tourpedia: a web application for sentiment visualization in tourism domain. *Proceedings of The OpeNER Workshop in The 9th edition of the Language Resources and Evaluation Conference (LREC 2014)*, pages 18–21, 2014.